大数据工程师的Python快速入门书
Scrapy、Beautiful Soup 网络爬虫实战
Mechanize、Selenium 模拟浏览器实战



Web Scraping with Python

Python 网络爬虫实战

胡松涛 编著



清华大学出版社



Python 网络爬虫实战

胡松涛 编著

清華大學出版社
北 京

本书从 Python 的安装开始，详细讲解了 Python 从简单程序延伸到 Python 网络爬虫的全过程。本书从实战出发，根据不同的需求选取不同的爬虫，有针对性地讲解了几种 Python 网络爬虫。

本书共 8 章，涵盖的内容有 Python 语言的基本语法、Python 常用 IDE 的使用、Python 第三方模块的导入使用、Python 爬虫常用模块、Scrapy 爬虫、Beautiful Soup 爬虫、Mechanize 模拟浏览器和 Selenium 模拟浏览器。本书所有源代码已上传网盘供读者下载。

本书内容丰富，实例典型，实用性强。适合 Python 网络爬虫初学者、数据分析与挖掘技术初学者，以及高校及培训学校相关专业的师生阅读。

图书在版编目（CIP）数据

Python 网络爬虫实战 / 胡松涛编著. — 北京：清华大学出版社，2017
ISBN 978-7-302-45787-9

I. ①P… II. ①胡… III. ①软件工具－程序设计 IV. ①TP311.56

中国版本图书馆 CIP 数据核字（2016）第 290543 号

责任编辑：夏非彼
封面设计：王　翔
责任校对：闫秀华
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张**：19　　**字　　数**：486 千字
版　　次：2017 年 1 月第 1 版　　**印　　次**：2017 年 1 月第 1 次印刷
印　　数：1~3000
定　　价：59.00 元

产品编号：071193-01

前　言

计算机技术飞速发展，人们对计算机使用技能的要求也越来越高。在编写软件时，大家既希望有超高的效率，又希望这门语言简单易用。这种鱼与熊掌皆得的要求的确很高，Python编程语言恰好符合这么苛刻的要求。

Python 的执行效率仅比效率之王 C 略差一筹，在简单易用方面 Python 也名列三甲。可以说 Python 在效率和简单之间达到了平衡。另外，Python 还是一门胶水语言，可以将其他编程语言的优点融合在一起，达到 1+1>2 的效果。这也是 Python 如今使用人数越来越多的原因。

Python 语言发展迅速，在各行各业都发挥独特的作用。在各大企业、学校、机关都运行着 Python 明星程序。但就个人而言，运用 Python 最多的还是网络爬虫（这里的爬虫仅涉及从网页提取数据，不涉及深度、广度算法爬虫搜索）。在网络上经常更新的数据，无须每次都打开网页浏览，使用爬虫程序，一键获取数据，下载保存后分析。考虑到 Python 爬虫在网络上的资料虽多，但大多都不成系统，难以提供系统有效的学习。因此笔者抛砖引玉，编写了这本有关 Python 网络爬虫的书，以供读者学习参考。

Python 简单易学，Python 爬虫也不复杂。只需要了解了 Python 的基本操作即可自行编写。本书中介绍了几种不同类型的 Python 爬虫，可以针对不同情况的站点进行数据收集。

本书特色

- 附带全部源代码

为了便于读者理解本书内容，作者已将全部的源代码上传到网络，供读者下载使用。读者通过代码学习开发思路，精简优化代码。

- 涵盖了 Linux&Windows 上模块的安装配置

本书包含了Python模块源的配置、模块的安装，以及常用IDE的使用。

- 实战实例

通过常用的实例，详细说明网络爬虫的编写过程。

本书结构

本书共 8 章，前面 4 章简单地介绍了 Python 的基本用法和简单 Python 程序的编写。第 5 章的 Scrapy 爬虫框架主要针对一般无须登录的网站，在爬取大量数据时使用 Scrapy 会很方便。第 6 章的 Beautiful Soup 爬虫可以算作爬虫的“个人版”。Beautiful Soup 爬虫主要针对一些爬取数据比较少的，结构简单的网站。第 7 章的 Mechanize 模块，主要功能是模拟浏览器。它的作用主要是针对那些需要登录验证的网站。第 8 章的 Selenium 模块，主要功能也是模拟浏览器，它的作用主要是针对 JavaScript 返回数据的网站。

本书读者与作者

- Python 网络爬虫初学者
- 数据分析与挖掘技术初学者
- 高校和培训学校相关专业的师生

本书由胡松涛主笔，其他参与编写的有宋士伟、张倩、彭霁、杨旺功、邹瑛、王铁民、殷龙、李春城、张兴瑜、刘祥淼、李柯泉、林龙、赵殿华、牛晓云。

本书代码下载

本书代码下载地址（注意数字和字母大小写）为 http://pan.baidu.com/s/1miTmq5y。如果下载有问题，请电子邮件联系booksaga@163.com，邮件主题为“网络爬虫代码”。

编　者

2016 年 11 月

目　录

第 1 章

◂Python环境配置▸

为什么选择 Python 来写网络爬虫？

众所周知 Python 的速度并不是最快的，比不上 Java，比不上 C++，更比不上传说中的速度效率之王 C 了。学习资料的完备也不在三甲之内，市面上讲解 C&C++的书籍绝对是 Python 的几倍甚至几十倍。使用的人数也不是最多，比不上 Java、C、C++。

那么，为什么会选择 Python？

首先是它简单易学。简单到没有学过任何编程语言的人稍微看下资料，再看几个示例就可以编写出可用的程序；其次它是一门解释型编程语言，编写完毕后可直接执行，无须编译，发现 Bug 后立即修改，省下了无数的编译时间；还有它的代码重用性高，可以把包含某个功能的程序当成模块代入其他程序中使用，因而 Python 的模块库庞大到恐怖，几乎是无所不包；最后就是因为它的跨平台性，几乎所有的 Python 程序，都可以不加修改地运行在不同的操作平台，都能得到同样的结果。这么多的优点都集中在这个语言中，因此最好的选择就是 Python。

1.1 Python 简介

了解一门语言，我们先从它的历史说起。Python 的应用越来越广泛，它最初是用来做什么用的，之后又如何发展的，了解这些，我们就更能了解 Python。

1.1.1 Python 的历史由来

Python 是一种开源的面向对象的脚本语言，它起源于 1989 年末，当时，CWI（阿姆斯特丹国家数学和计算机科学研究所）的研究员 Guido van Rossum 需要一种高级脚本编程语言，为其研究小组的 Amoeba 分布式操作系统执行管理任务。为创建新语言，他从高级数学语言 ABC（ALL BASIC CODE）汲取了大量语法，并从系统编程语言 Modula-3 借鉴了错误处理机制。Van Rossum 把这种新的语言命名为 Python（大蟒蛇）——来源于 BBC 当时正在热播的喜剧连续剧 Monty Python。

ABC 是由 Guido 参加设计的一种教学语言。就 Guido 本人看来，ABC 这种语言非常优

美和强大，是专门为非专业程序员设计的。但是 ABC 语言并没有成功，究其原因，Guido 认为是非开放造成的。Guido 决心在 Python 中避免这一错误。同时，他还想实现在 ABC 中闪现过但未曾实现的东西。

就这样，Python 在 Guido 手中诞生了。可以说，Python 是从 ABC 发展起来，并且结合了 Unix shell 和 C 的习惯。Python 源代码遵循 GPL（GNU General Public License）协议。所以任何个人用户都可以免费使用。

1.1.2 Python 的现状

Python 于 1991 年初公开发行，由于功能强大和采用开源方式发行，Python 发展得很快，用户越来越多，形成了一个强大的社区力量。2001 年，Python 的核心开发团队移师 Digital Creations 公司，该公司是 Zope（一个用 Python 编写的 Web 应用服务器）的创始者。大家可到 http://www.python.org/上了解最新的 Python 动态和资料 。

如今，Python 已经成为最受欢迎的程序设计语言之一。2011 年 1 月，它被 TIOBE 编程语言排行榜评为 2010 年度语言。自从 2004 年以后，Python 的使用率是呈线性增长 。

1.1.3 Python 的应用

Python 应用广泛，特别适用与以下几个方面。

- 系统编程：提供 API（Application Programming Interface，应用程序编程接口），能方便地进行系统维护和管理，Linux 下标志性语言之一，是很多系统管理员理想的编程工具 。
- 图形处理：有 PIL、Tkinter 等图形库支持，能方便进行图形处理。
- 数学处理：NumPy 扩展提供大量与许多标准数学库的接口。
- 文本处理：Python 提供的 re 模块能支持正则表达式，还提供 SGML、XML 分析模块，许多程序员利用 Python 进行 XML 程序的开发。
- 数据库编程：程序员可通过遵循 Python DB-API（数据库应用程序编程接口）规范的模块与 Microsoft SQL Server、Oracle、Sybase、DB2、MySQL、SQLite 等数据库通信。Python 自带有一个 Gadfly 模块，提供了一个完整的 SQL 环境。
- 网络编程：提供丰富的模块支持 sockets 编程，能方便快速地开发分布式应用程序。很多大规模软件开发计划，例如 Zope、Mnet 及 BitTorrent、Google 都在广泛地使用它。
- Web 编程：应用的开发语言，支持最新的 XML 技术。
- 多媒体应用：Python 的 PyOpenGL 模块封装了 OpenGL 应用程序编程接口，能进行二维和三维图像处理。PyGame 模块可用于编写游戏软件。
- PYMO 引擎：PYMO 全称为 Python Memories Off，是一款运行于 Symbian S60V3、Symbian3、S60V5、Symbian3、Android 系统上的 AVG 游戏引擎。因其基于 Python2.0 平台开发，并且适用于创建秋之回忆（memories off）风格的 AVG 游戏，

故命名为 PYMO。

不只个人用户推崇 Python，企业用户也对 Python 青睐有加，以下是明星企业的应用项目：

- Reddit：社交分享网站，最早用 Lisp 开发，在 2005 年转为 python。
- Dropbox：文件分享服务。
- 豆瓣网：图书、唱片、电影等文化产品的资料数据库网站。
- Django：鼓励快速开发的 Web 应用框架。
- Fabric：用于管理成百上千台 Linux 主机的程序库。
- EVE：网络游戏 EVE 大量使用 Python 进行开发。
- Blender：以 C 与 Python 开发的开源 3D 绘图软件。
- BitTorrent：bt 下载软件客户端。
- Ubuntu Software Center：Ubuntu 9.10 版本后自带的图形化包管理器。
- YUM：用于 RPM 兼容的 Linux 系统上的包管理器。
- Civilization IV：游戏《文明 4》。
- Battlefield 2：游戏《战地 2》。
- Google：谷歌在很多项目中用 python 作为网络应用的后端，如 Google Groups、Gmail、Google Maps 等，Google App Engine 支持 python 作为开发语言。
- NASA：美国宇航局，从 1994 年起把 python 作为主要开发语言。
- Industrial Light & Magic：工业光魔，乔治·卢卡斯创立的电影特效公司。
- Yahoo! Groups：雅虎推出的群组交流平台。
- YouTube：视频分享网站，在某些功能上使用到 python。
- Cinema 4D：一套整合 3D 模型、动画与绘图的高级三维绘图软件，以其高速的运算和强大的渲染插件著称。
- Autodesk Maya：3D 建模软件，支持 python 作为脚本语言。
- gedit：Linux 平台的文本编辑器。
- GIMP：Linux 平台的图像处理软件。
- Minecraft: Pi Edition：游戏《Minecraft》的树莓派版本。
- MySQL Workbench：可视化数据库管理工具。
- Digg：社交新闻分享网站。
- Mozilla：为支持和领导开源的 Mozilla 项目而设立的一个非营利组织。
- Quora：社交问答网站。
- Path：私密社交应用。
- Pinterest：图片社交分享网站。
- SlideShare：幻灯片存储、展示、分享的网站。
- Yelp：美国商户点评网站。
- Slide：社交游戏/应用开发公司，被谷歌收购。

还有很多企业级的应用这里就不一一列举了。Python 适用于不同的场合、不同的人群，

是适应性非常强的一门语言。

1.2 Python 开发环境配置

Python 在 PC 三大主流平台（Windows、Linux 和 OS X）都可使用。在这里只讲解 Windows 和 Linux 下的开发环境配置。Windows 平台以 Windows 7 为例，Linux 平台以 Debian 8 系统为例。Python 目前主要有两个版本，Python 2 和 Python 3。目前，Python 2 的最终版本是 Python 2.7.11，Python 3 的最终版本是 Python 3.5.1。Python 3 虽然功能更加强大，但暂时 Python 2 的使用人数更多，本书中全部选择 Python 2.7 为例。

1.2.1 Windows 下安装 Python

（1）打开 Chrome 浏览器，在地址栏输入 Python 官网地址 www.python.org，如图 1-1 所示。

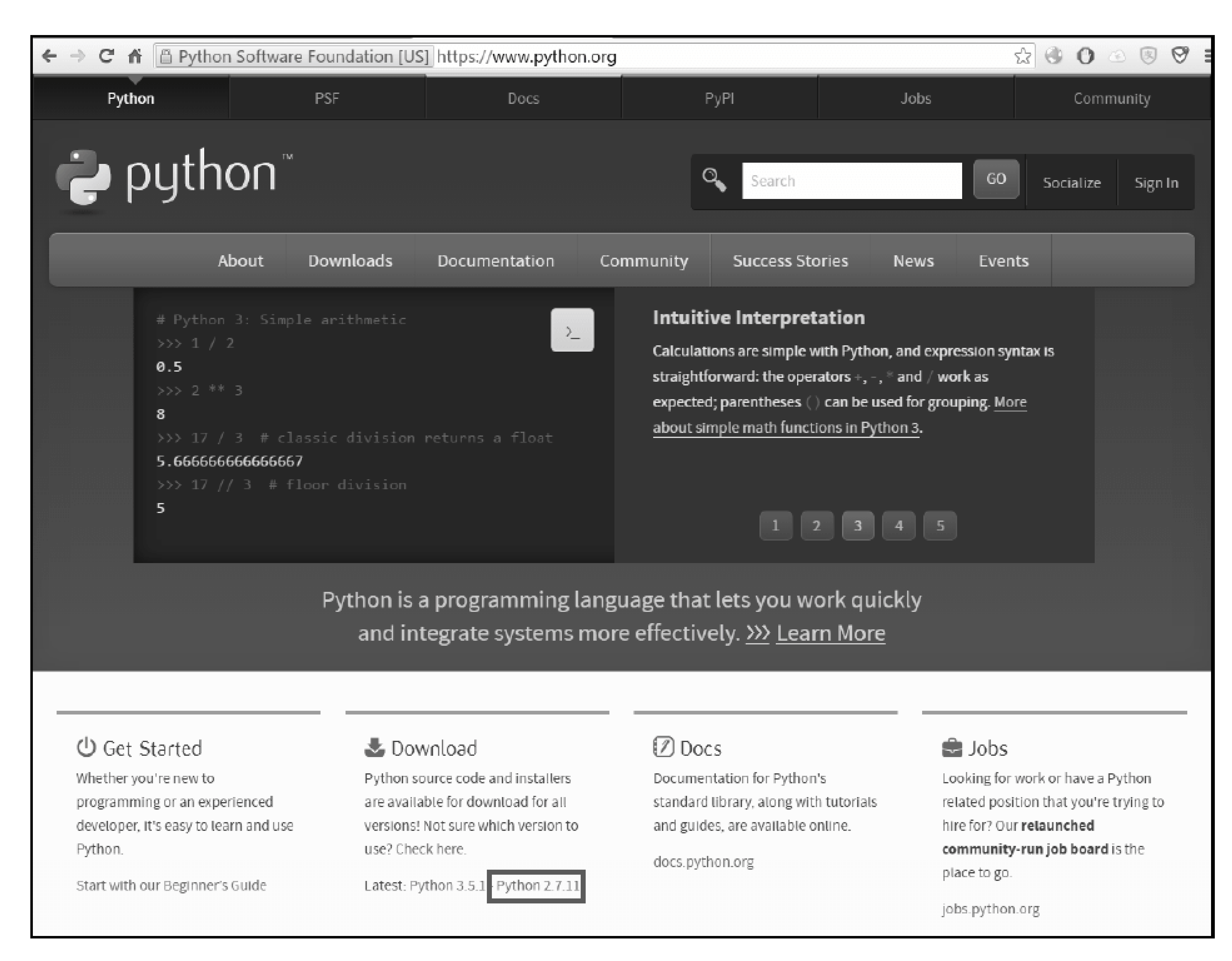
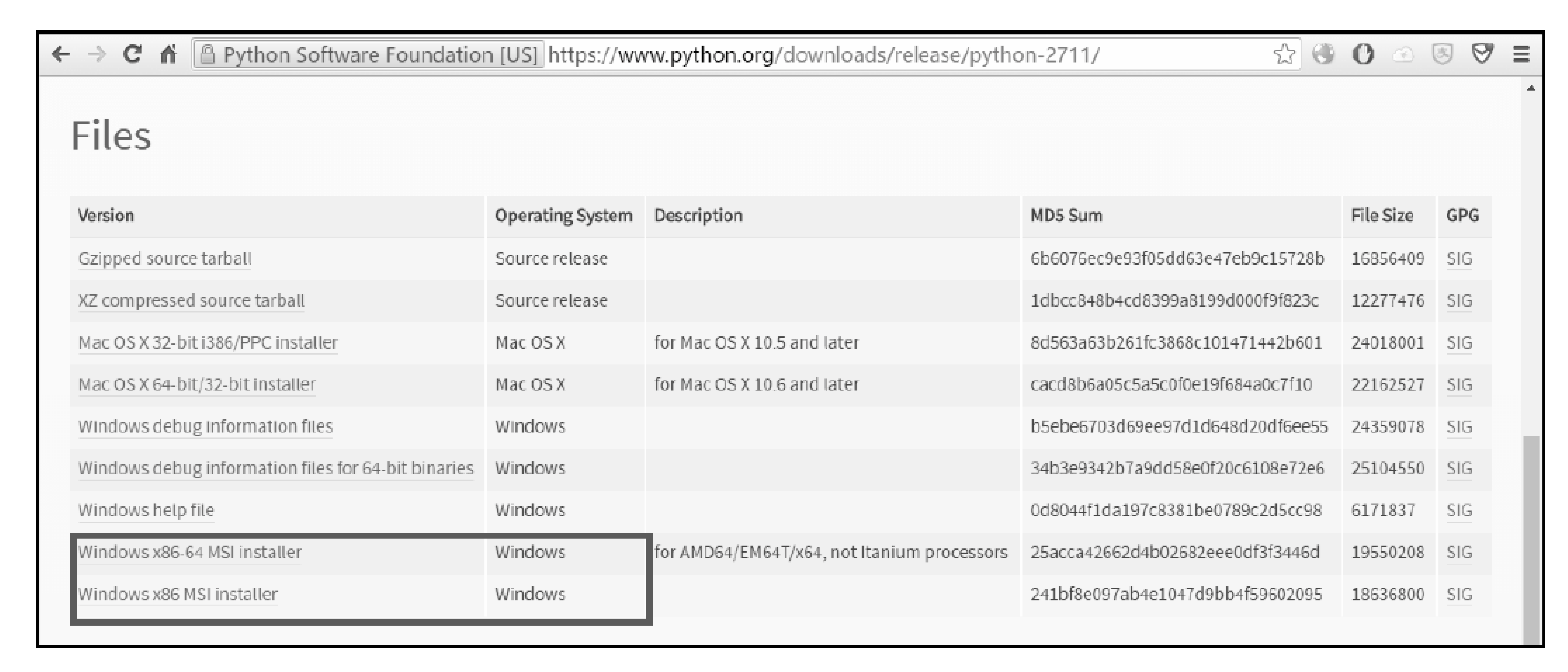


图 1-1　Python 官网

（2）单击 Python 2.7.11，进入 Python 2.7.11 的下载页面，如图 1-2 所示。



Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		6b6076ec9e93f05dd63e47eb9c15728b	16856409	SIG
XZ compressed source tarball	Source release		1dbcc848b4cd8399a8199d000f9f823c	12277476	SIG
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	8d563a63b261fc3868c101471442b601	24018001	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	cacd8b6a05c5a5c0f0e19f684a0c7f10	22162527	SIG
Windows debug information files	Windows		b5ebe6703d69ee97d1d648d20df6ee55	24359078	SIG
Windows debug information files for 64-bit binaries	Windows		34b3e9342b7a9dd58e0f20c6108e72e6	25104550	SIG
Windows help file	Windows		0d8044f1da197c8381be0789c2d5cc98	6171837	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	25acca42662d4b02682eee0df3f3446d	19550208	SIG
Windows x86 MSI installer	Windows		241bf8e097ab4e1047d9bb4f59602095	18636800	SIG

图 1-2　Python 下载

（3）按照安装的 Windows 系统选择下载的安装文件。示例系统是 Windows 7 64 位版本，所以在此下载的是 Windows x86-64 installer。

（4）下载完毕，得到安装文件 python-2.7.11.amd64.msi。双击该文件图标，开始安装 Python 2.7，如图 1-3 所示。



图 1-3　安装 Python

（5）单击 Next 按钮，设置 Python 安装路径，如图 1-4 所示。



图 1-4　设置 Python 安装路径

（6）选择或者填入 Python 的安装路径后，单击 Next 按钮，进入 Python 组件设置，如图 1-5 所示。

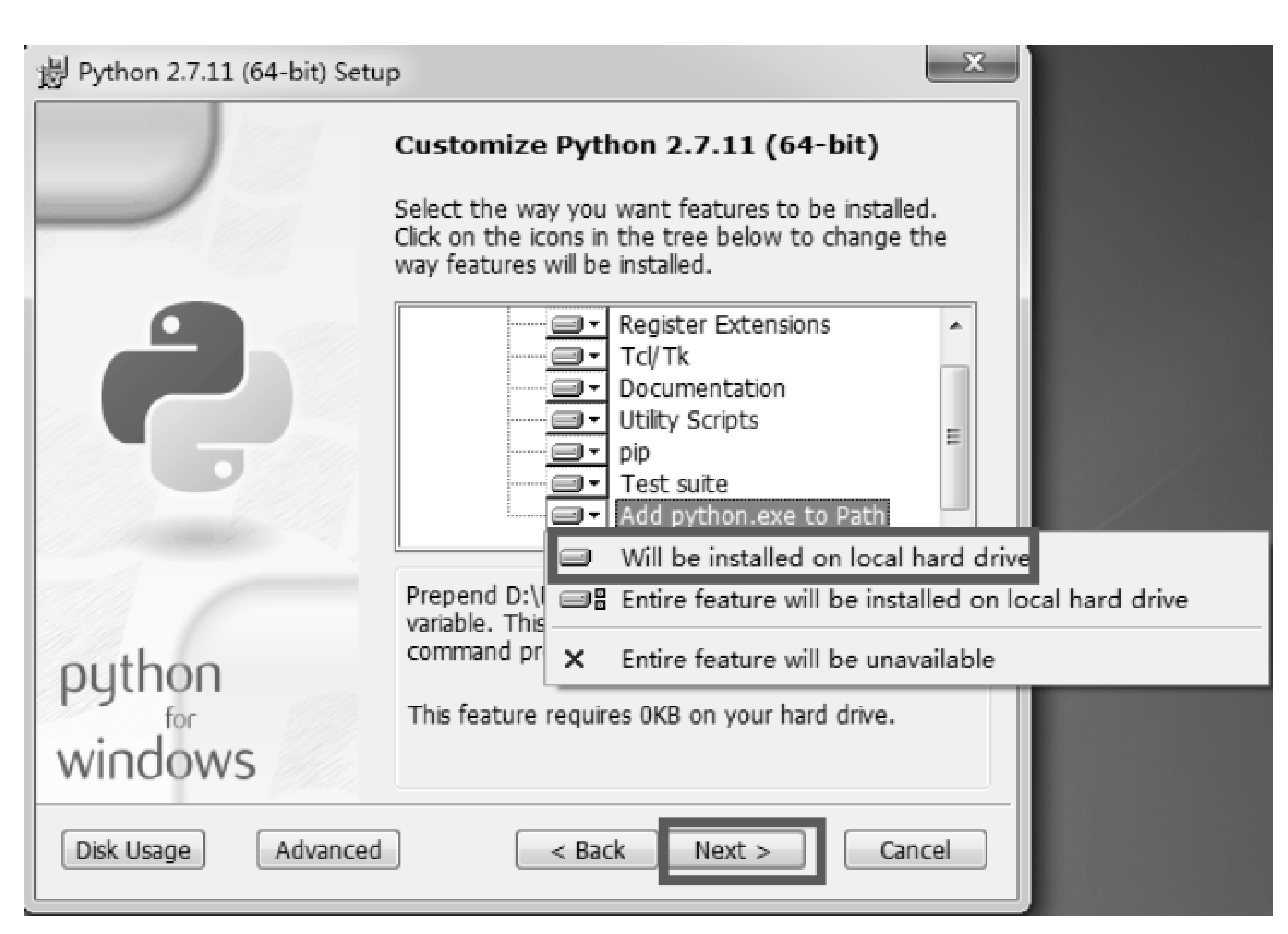


图 1-5　Python 组件选择

默认情况下 Add python.exe to Path 这个组件是未选择的，它的作用是将 Python 的路径加入系统环境中。请将它选择上，单击 Next 按钮，开始安装 Python，如图 1-6 所示。



图 1-6　安装 Python

（7）单击 Finish 按钮，整个安装程序完毕。验证 Python 是否安装成功。单击桌面左下角的“开始”菜单，在地址栏输入 cmd.exe 后按 Enter 键，打开 Windows 系统命令行程序，如图 1-7 所示。



图 1-7　打开系统命令行工具 cmd.exe

（8）执行命令，验证 Python，如图 1-8 所示。



图 1-8　验证 Python

好了，由此可见 Python 已安装成功，并已将路径添加到环境变量。单击桌面左下的“开始”|“所有程序”菜单，单击 Python 2.7 菜单，就可以看到 Python 的菜单，如图 1-9 所示。



图 1-9　Python 2.7 菜单



在安装 Python 的同时也安装了 Python 自带的 IDE——IDLE 和本地的模块说明文档。这个文档的说明很详细，一般只用看这个文档就足够了。

至此 Python 已在 Windows 上安装验证成功，可以愉快地使用 Python 了。

1.2.2　Windows 下安装配置 pip

上文中说过，Python 有几乎无限的第三方模块。如何安装这些第三方模块呢？这里就不得不说到 easy_install 和 pip 了。

easy_install 和 pip 都是 Python 的模块安装工具，有点类似于 Debian 系统的 apt-get，Fedora 系统中的 yum，或者是 Windows 系统中的 QQ 软件管理器，都是一键安装软件工具，所不同的是它们只负责安装 Python 模块。老版本中的 Python 只有 easy_install。pip 可以认为是 easy_install 的高级版本，所以 pip 和 easy_install 任选其一都可以，个人建议使用 pip。在安装 Python 时已经选择了 pip 组件（如图 1-5 所示），就无须再次安装 pip 了，直接开始配置即可。

因为 pip 的服务器，也就是安装源在国外，基于国内糟糕的网络环境，使用 pip 安装 python 第三方模块将是一个很痛苦的过程。好在还有变通的方法，在国内也有 pip 的镜像源。只需要在 pip 的配置文件中将 pip 的安装源指向国内的服务器，这个问题就解决了。

根据 pip 的指南，Windows 中 pip 的配置文件是%HOME%/pip/pip.ini（具体到当前环境，Windows 的当前用户是 king，所以配置文件位置就是 C:\Users\king\pip\pip.ini）。默认情况下 pip 文件夹和 pip.ini 文件都未被创建，需要自行创建。按照指南创建好文件夹和文件后，修改 pip.ini 文件，如图 1-10 所示。



图 1-10　修改 pip.ini

图 1-10 中准备了 3 个 pip 源，任选其一都可以。选择的方法就是在不需要的源地址前面加上#符号。下面来验证一下修改源地址是否成功，执行命令：

```
python -m pip install -upgrade pip
```

此命令的作用是更新 pip 源，结果如图 1-11 所示。

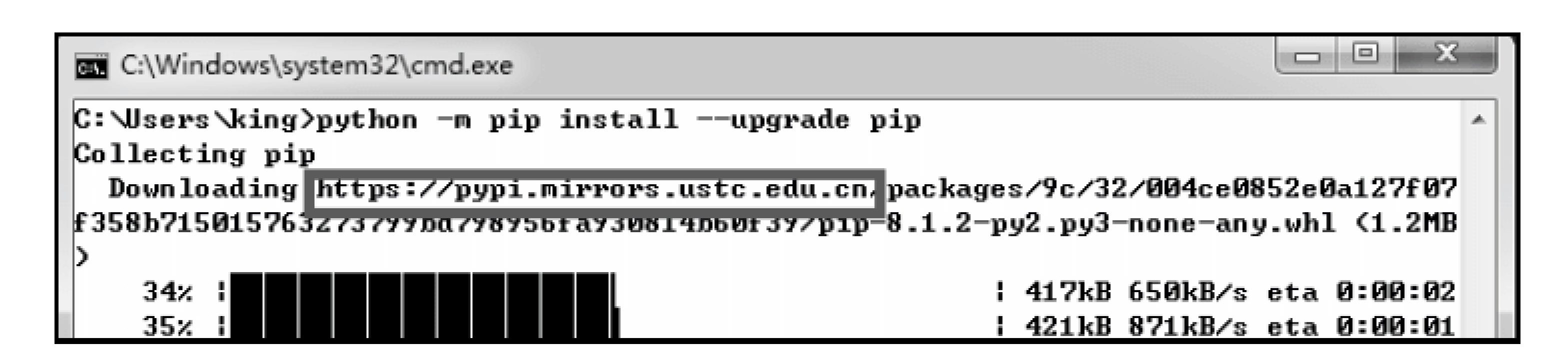


图 1-11　更新 pip 源

可以看出，配置文件中的新源已经起作用了。测试一下 pip。单击桌面左下角的“开始”菜单，在地址栏中输入 cmd.exe 后按 Enter 键，打开 Windows 系统命令行程序。执行命令，如图 1-12 所示。

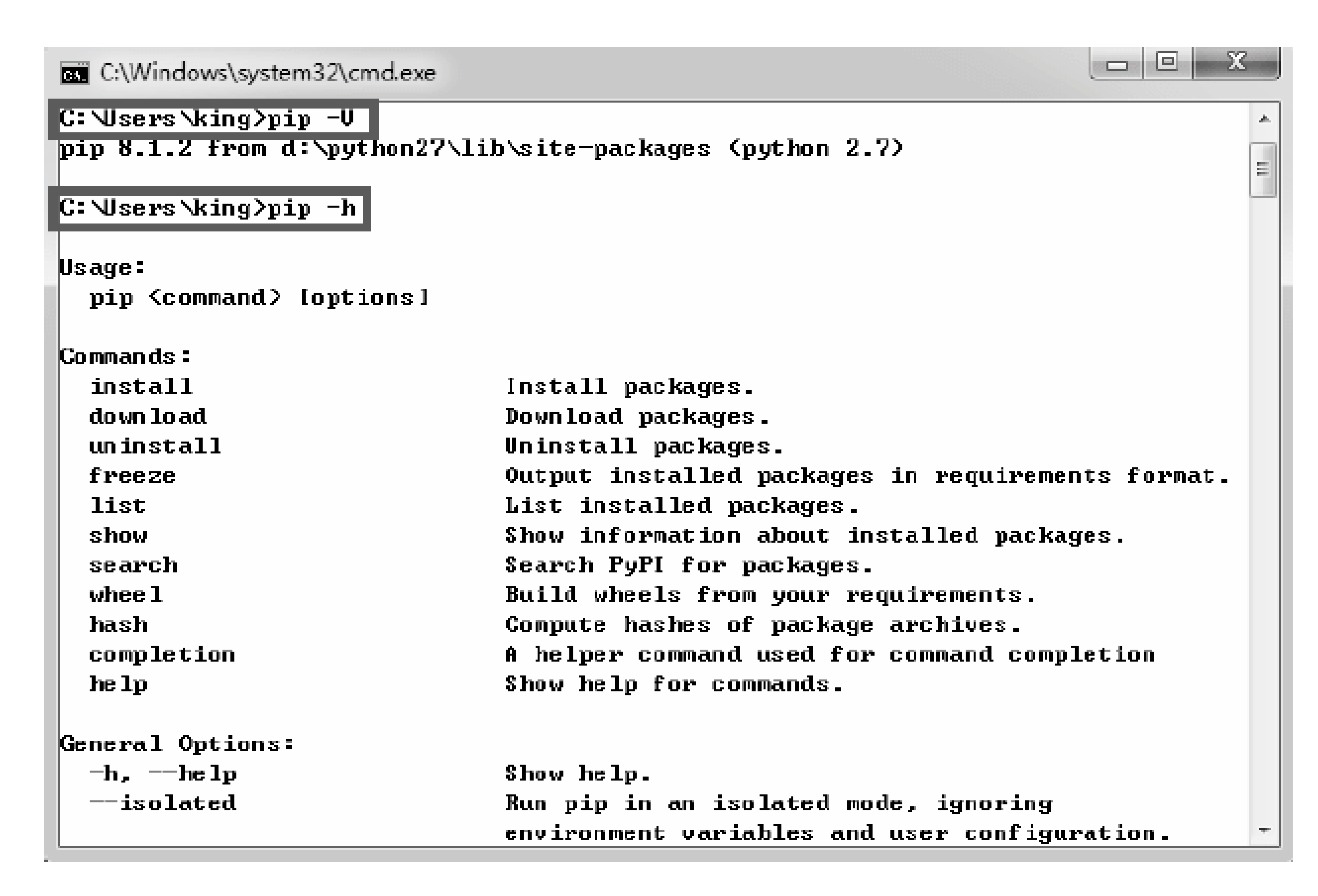


图 1-12　测试 pip

到此 pip 已完全配置完毕。

1.2.3　Linux 下安装 Python

即使同一个版本的 Linux，也因为不同的桌面环境而变得复杂起来。丰富的软件，是 Linux 的成功之处。没有统一的标准，却是 Linux 不能做大做强的关键所在。为避免桌面环境的差异，这里统一使用 Putty 来连接 Linux。对于 Python 而言，任意版本的 Linux 都没有任何的区别。这里演示用的 Linux 系统为 Debian 8.0，IP 地址为 192.168.2.80。下面，先用 Putty 连接这个 Linux 机器。

（1）双击 Putty 图标，打开 Putty.exe，填入 IP 地址和端口信息，如图 1-13 所示。

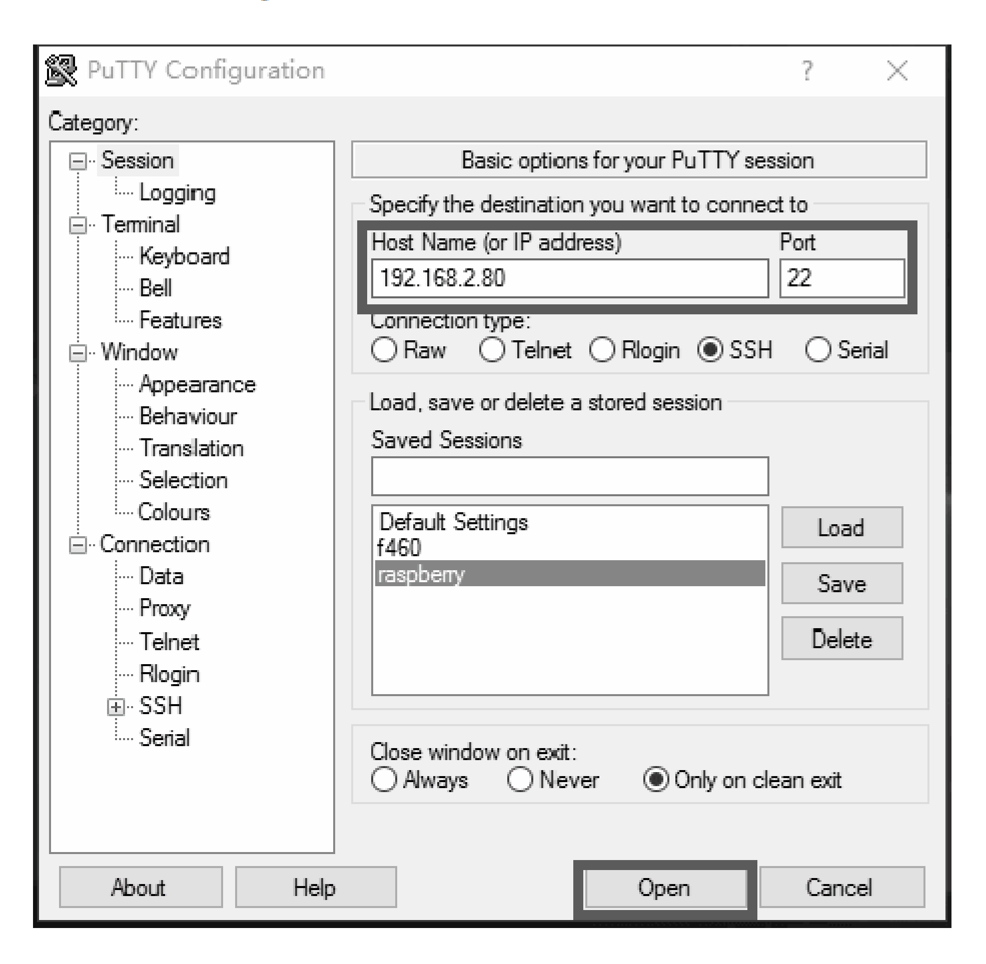


图 1-13　Putty 连接设置

（2）单击 Open 按钮，第一次使用 Putty 登录 Linux 会有一个安全警告提示，如图 1-14 所示。



图 1-14　Putty 安全警告

（3）单击“是(Y)”按钮，进入了 Linux 的登录界面，如图 1-15 所示。

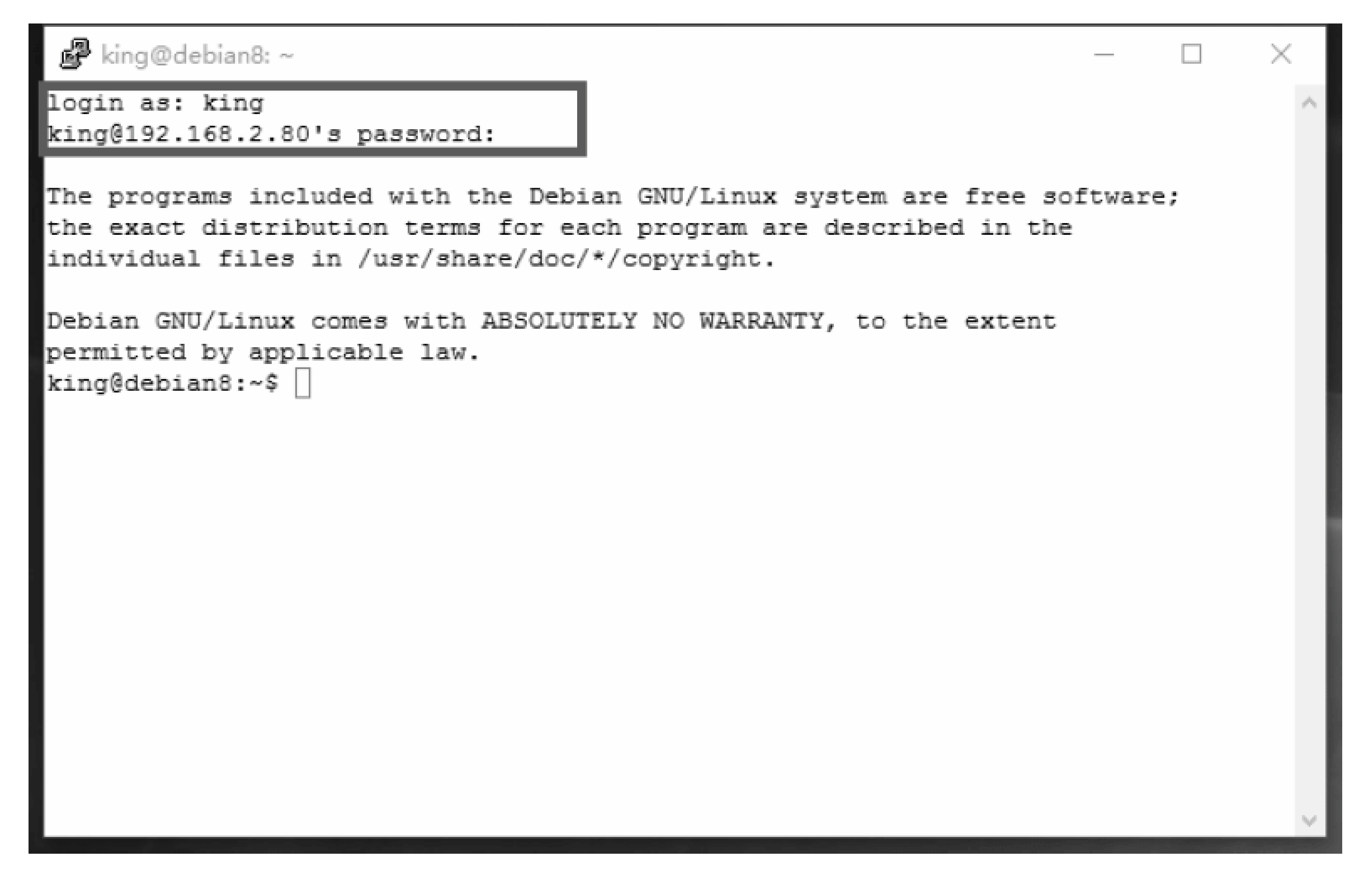


图 1-15　登录 Linux

（4）输入用户名和用户密码后（用户密码不回显），登录到了 Linux。

Debian Linux 默认安装了 Python 2 和 Python 3（几乎所有的 Linux 发行版本都默认安装了 Python）。Python 命令默认指向 Python 2.7，验证一下 Python 的路径，执行命令：

```
whereis python
ls -l /usr/bin/python
ls -l /usr/bin/python3
```

执行的结果如图 1-16 所示。

```
king@debian8: ~
login as: king
king@192.168.2.80's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed May 18 11:30:58 2016 from 192.168.2.99
king@debian8:~$ whereis python
python: /usr/bin/python2.7 /usr/bin/python /usr/bin/python3.4m /usr/bin/python3.
4 /usr/lib/python2.6 /usr/lib/python2.7 /usr/lib/python3.4 /etc/python2.7 /etc/p
ython /etc/python3.4 /usr/local/lib/python2.7 /usr/local/lib/python3.4 /usr/incl
ude/python2.7 /usr/share/python /usr/share/man/man1/python.1.gz
king@debian8:~$ ls -l /usr/bin/python
lrwxrwxrwx 1 root root 9 5月  13 01:00 /usr/bin/python -> python2.7
king@debian8:~$ ls -l /usr/bin/python3
lrwxrwxrwx 1 root root 9 5月  13 01:00 /usr/bin/python3 -> python3.4
king@debian8:~$
```

图 1-16　查看 Python 路径

再来看看 Python 的版本信息，执行命令：

```
python2 -V
python3 -V
```

执行的结果如图 1-17 所示。



图 1-17　Python 版本信息

从图 1-17 中可以看出，Linux 上安装的 Python 的版本与官网上的最新版本（Python 3.5.1-Python 2.7.11）是不同的。这是正常现象，一般来说 Debian Linux 会使用软件的最稳定版本，而 Ubuntu Linux 会使用软件的最新版本。

1.2.4　Linux 下安装配置 pip

如同 Windows 中的 Python 一样，Linux 中的 Python 同样需要一个模块安装的管理工具，可以是 easy_install，也可以是 pip。遗憾的是多数 Linux 版本并没有默认安装这个管理工具（Debian 可以用 apt-get 安装大部分的 Python 第三方模块，只有极少数的模块不能使用 apt-get 安装），所以得自己安装它了。

从 Debian Linux 中安装 pip，执行命令：

```
su -
apt-get install python-pip
```

执行结果如图 1-18 所示。


```
king@debian8:~$ su -
密码:
root@debian8:~# apt-get install python-pip
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列软件包是自动安装的并且现在不需要了:
  libasn1-8-heimdal libgssapi3-heimdal libhcrypto4-heimdal
  libheimbase1-heimdal libheimntlm0-heimdal libhx509-5-heimdal
  libkrb5-26-heimdal libroken18-heimdal libwind0-heimdal
使用'apt-get autoremove'来卸载它(它们)。
将会安装下列额外的软件包:
  build-essential dpkg-dev g++ g++-4.9 libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libdpkg-perl
  libfile-fcntllock-perl libstdc++-4.9-dev python-cffi python-colorama
  python-cryptography python-distlib python-html5lib python-ndg-httpsclient
  python-openssl python-ply python-pyasn1 python-pycparser python-requests
  python-setuptools python-urllib3 python-wheel
建议安装的软件包:
  debian-keyring g++-multilib g++-4.9-multilib gcc-4.9-doc libstdc++6-4.9-dbg
  libstdc++-4.9-doc python-dev python-cryptography-doc
  python-cryptography-vectors python-genshi python-openssl-doc
  python-openssl-dbg python-ply-doc doc-base
推荐安装的软件包:
  python-dev-all
下列【新】软件包将被安装:
  build-essential dpkg-dev g++ g++-4.9 libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libdpkg-perl
  libfile-fcntllock-perl libstdc++-4.9-dev python-cffi python-colorama
  python-cryptography python-distlib python-html5lib python-ndg-httpsclient
  python-openssl python-pip python-ply python-pyasn1 python-pycparser
  python-requests python-setuptools python-urllib3 python-wheel
升级了 0 个软件包，新安装了 25 个软件包，要卸载 0 个软件包，有 2 个软件包未被升
级。
需要下载 26.6 MB 的软件包。
解压缩后会消耗掉 56.8 MB 的额外空间。
您希望继续执行吗？ [Y/n] y
```

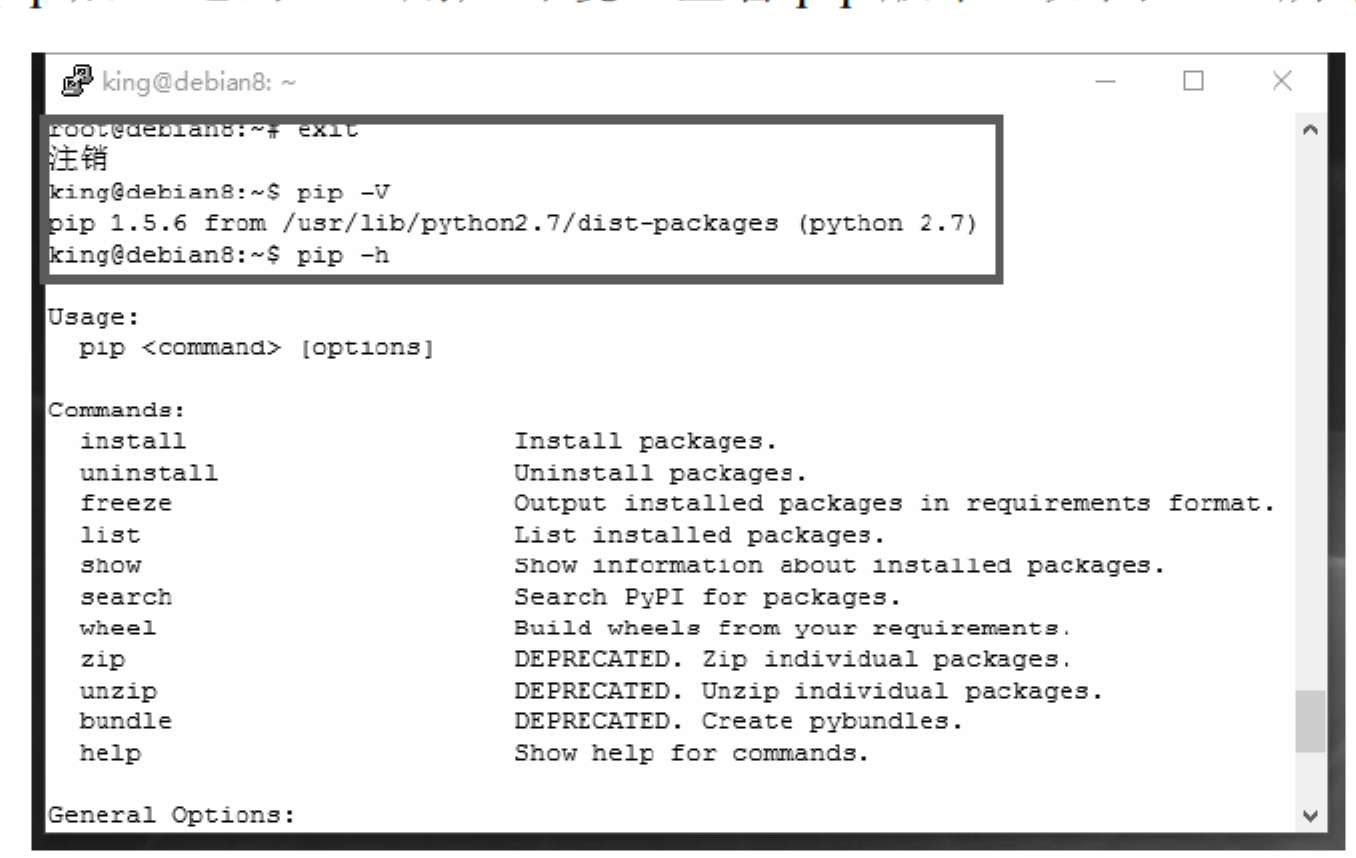
图 1-18 安装 pip

输入 su -命令后再输入系统 root 用户的登录密码。该命令的作用是使用 root 用户登录系统，并使用 root 用户的环境变量。apt-get install python-pip 作用是使用 apt-get 命令安装 python-pip 这个工具包。最后输入 y 确认执行命令，开始安装 python-pip。



Linux 下安装软件都必须有 root 权限，可以直接转换成 root 用户安装，也可以在 sudoers 里添加用户和权限。

安装 python-pip 后，退出 root 用户环境，查看 pip 版本，如图 1-19 所示。


```
root@debian8:~# exit
注销
king@debian8:~$ pip -V
pip 1.5.6 from /usr/lib/python2.7/dist-packages (python 2.7)
king@debian8:~$ pip -h

Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  search                      Search PyPI for packages.
  wheel                       Build wheels from your requirements.
  zip                         DEPRECATED. Zip individual packages.
  unzip                       DEPRECATED. Unzip individual packages.
  bundle                      DEPRECATED. Create pybundles.
  help                        Show help for commands.

General Options:
```

图 1-19 验证 pip

最后还要将 pip 的更新源改成国内源。根据 pip 的指南，在 Linux 下 pip 的配置文件是 $HOME/.pip/pip.conf，执行命令：

```
su -
cd
pwd
mkdir .pip
cd .pip
cat > pip.conf << EOF
[global]
index-url = https://pypi.mirrors.ustc.edu.cn/simple
#index-url = http://pypi.hustunique.com/simple
#index-url = http://pypi.douban.com/simple
EOF
cat pip.conf
exit
```

执行结果如图 1-20 所示。

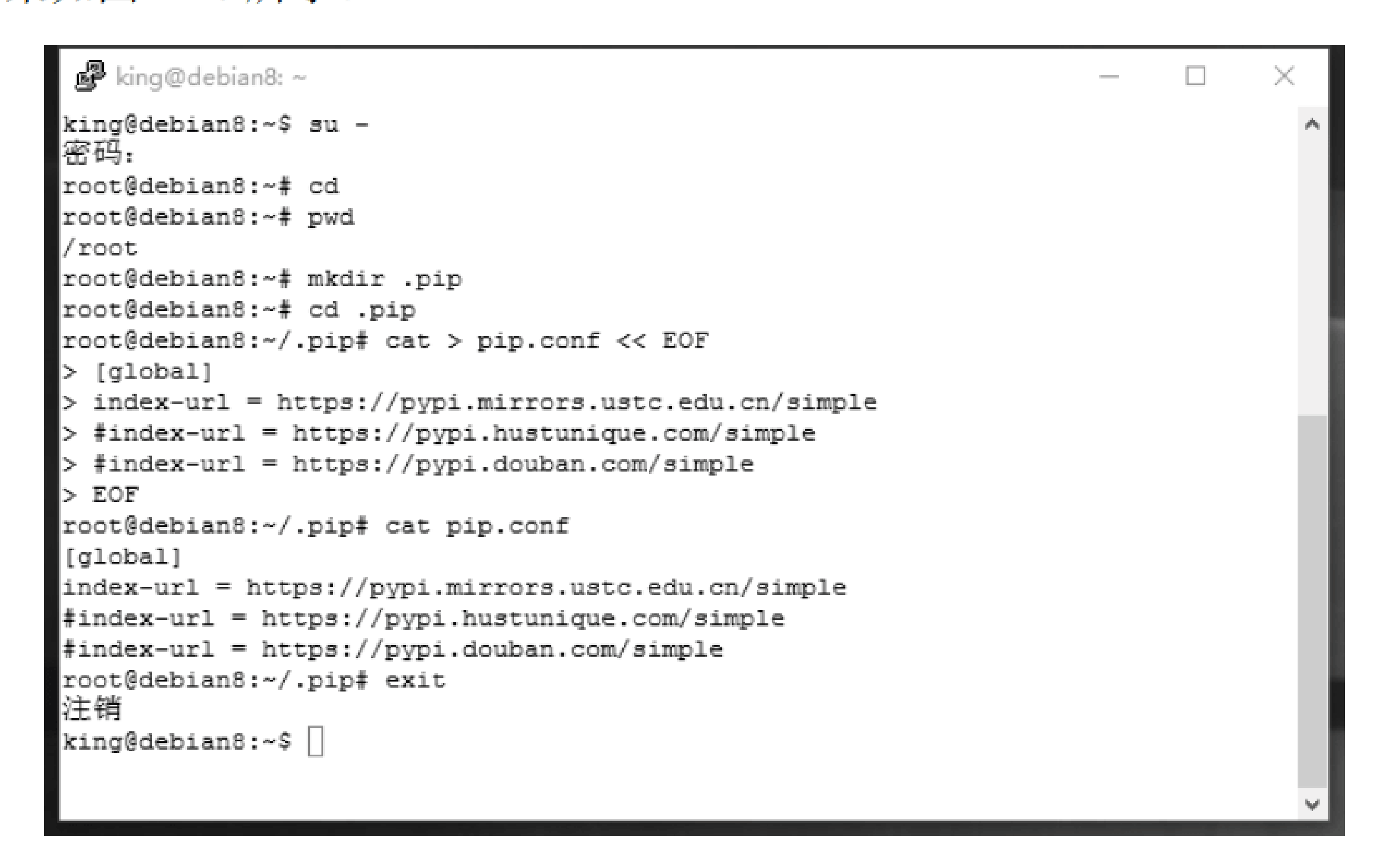


图 1-20　修改 pip.conf

验证一下修改源地址是否成功，执行命令：

```
su -
python -m pip install -upgrade pip
exit
```

结果如图 1-21 所示。

```
king@debian8: ~
king@debian8:~$ su -
密码:
root@debian8:~# python -m pip install --upgrade pip
Downloading/unpacking pip from https://pypi.mirrors.ustc.edu.cn/packages/9c/32/0
04ce0852e0a127f07f358b71501576327379 9bd798956fa930814b60f39/pip-8.1.2-py2.py3-no
ne-any.whl#md5=0570520434c5b600d89ec95393b2650b
  Downloading pip-8.1.2-py2.py3-none-any.whl (1.2MB): 1.2MB downloaded
Installing collected packages: pip
  Found existing installation: pip 1.5.6
    Not uninstalling pip at /usr/lib/python2.7/dist-packages, owned by OS
Successfully installed pip
Cleaning up...
root@debian8:~# exit
注销
king@debian8:~$
```

图 1-21　更新 pip 源

从图 1-21 可以看出 pip 源已经开始起作用了。下面来测试一下 pip，如图 1-22 所示。

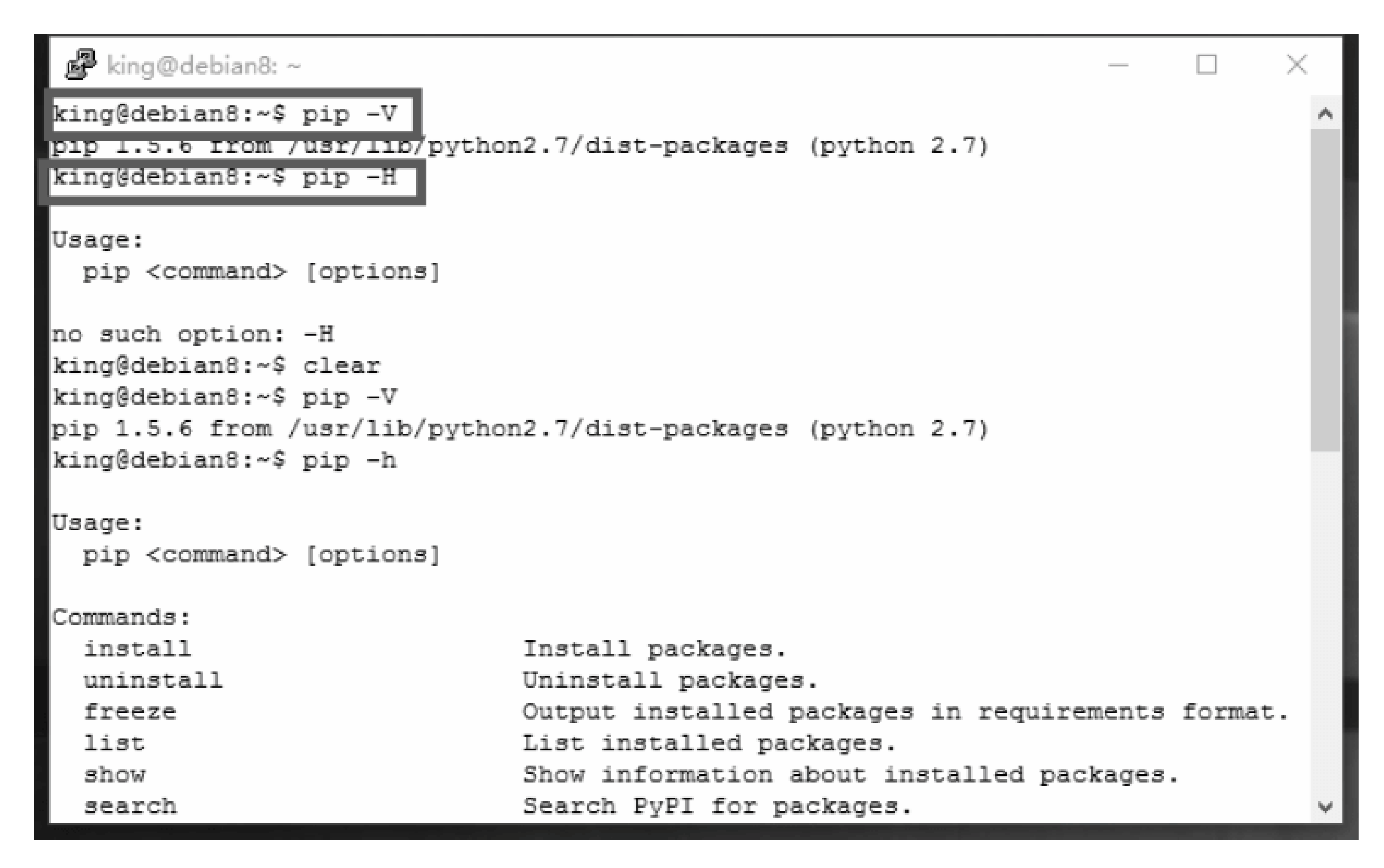


图 1-22　测试 pip

到此 pip 已完全配置完毕。和 Windows 下的 pip 不同，Linux 下的 pip 可以用 root 安装模块，也可以使用一般用户来安装模块。推荐使用 root 用户来安装，因为有些模块安装需要 root 特权，root 安装的模块一般用户都可以使用。

1.2.5　永远的开始：hello world

似乎所有的编程语言第一个程序都是 hello world。Python 也不能免俗，下面分别从 Windows 和 Linux 下创建 hello.py。

1．Windows 下创建 hello.py

（1）单击桌面左下的“开始”|“所有程序”菜单，单击 Python 2.7 菜单，单击 IDLE (Python GUI)菜单，如图 1-23 所示。



图 1-23　打开 IDLE

（2）此时打开的是 Python shell 交互界面，再单击 File|New File 菜单，如图 1-24 所示。

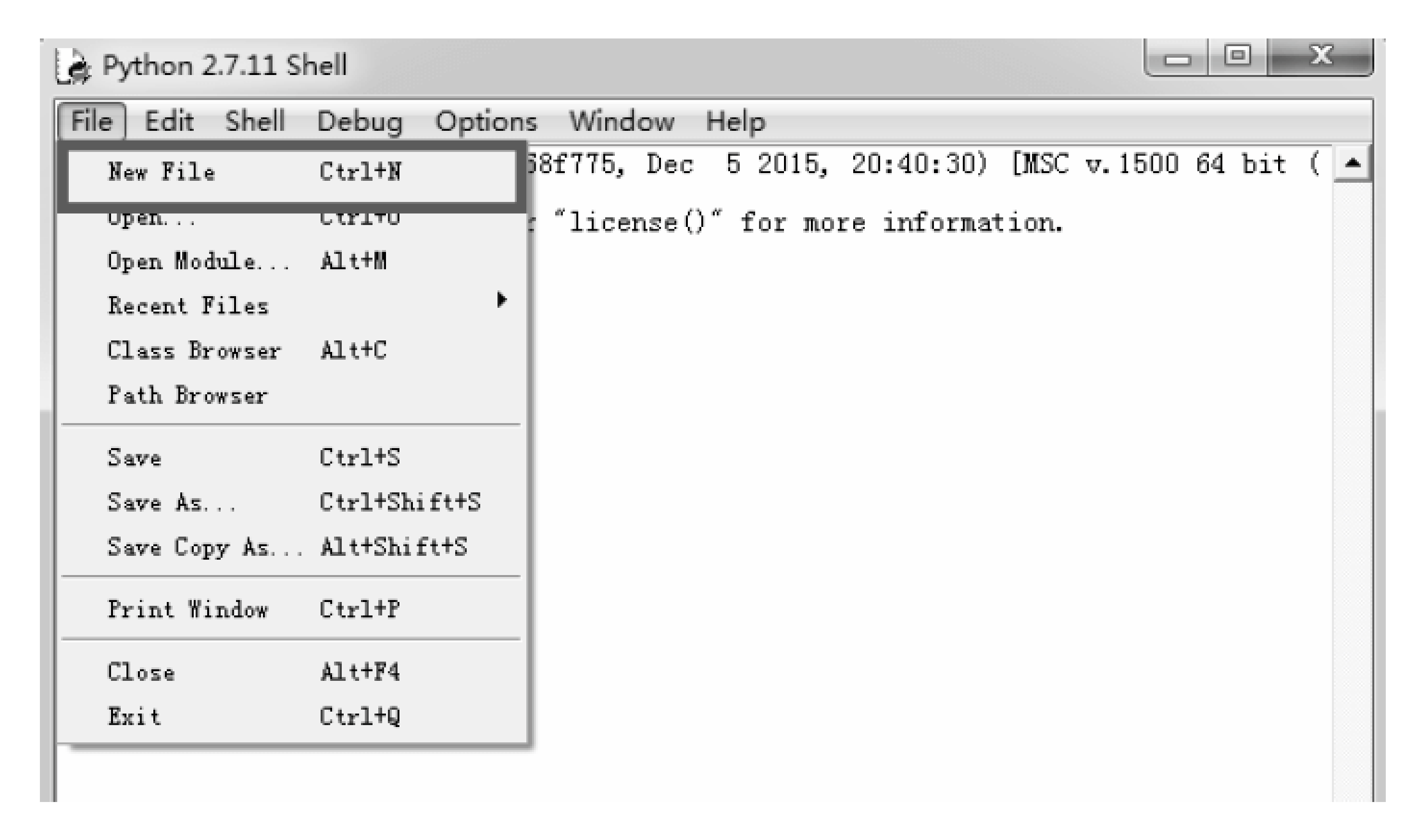


图 1-24　打开 IDE

（3）用 IDLE 的 IDE 打开了一个新文件，在此新文件中编辑 hello.py，如图 1-25 所示。

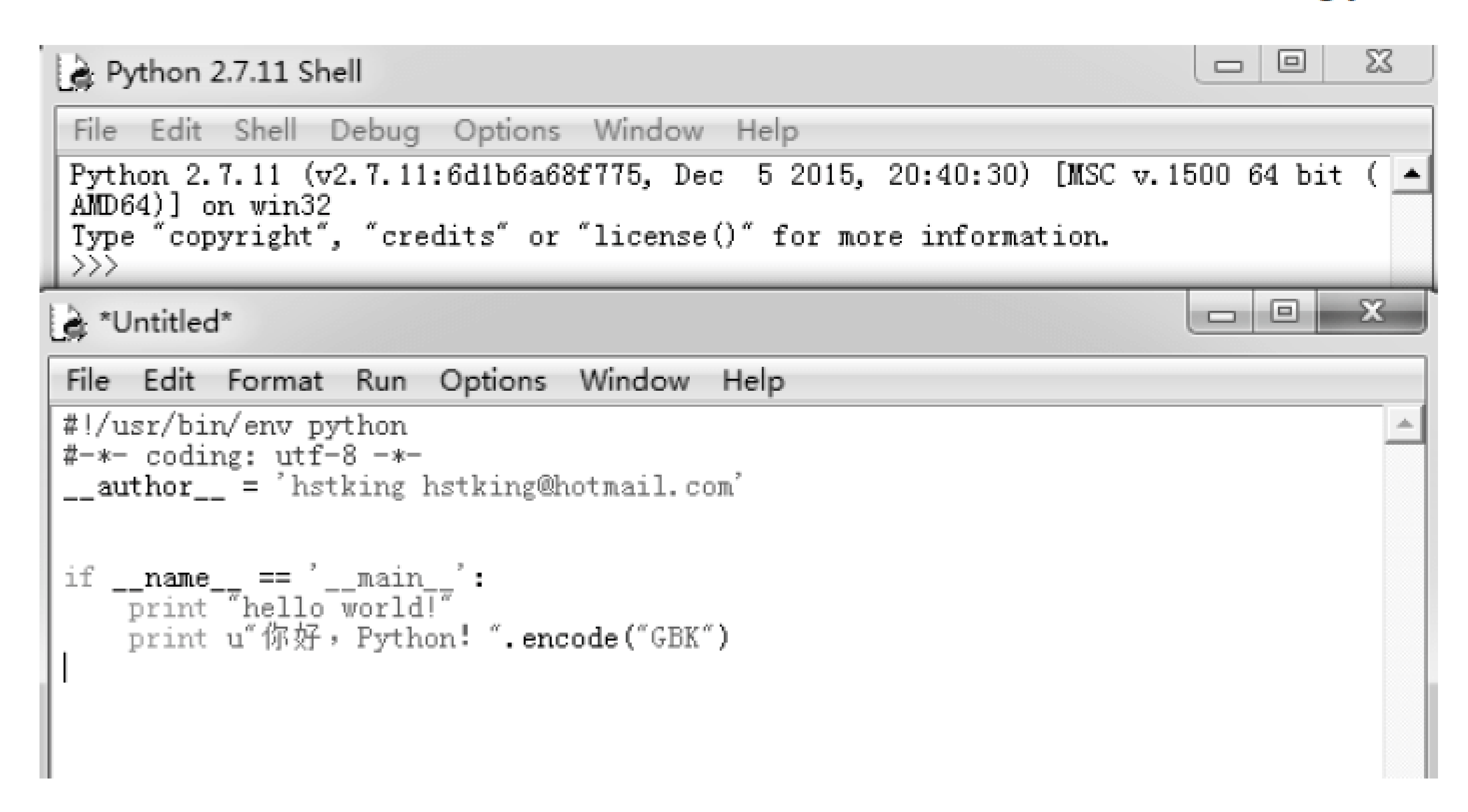


图 1-25　编辑 hello.py

（4）单击该 IDE 的 File|Save As ...菜单，将已编辑好的代码保存，如图 1-26 所示。

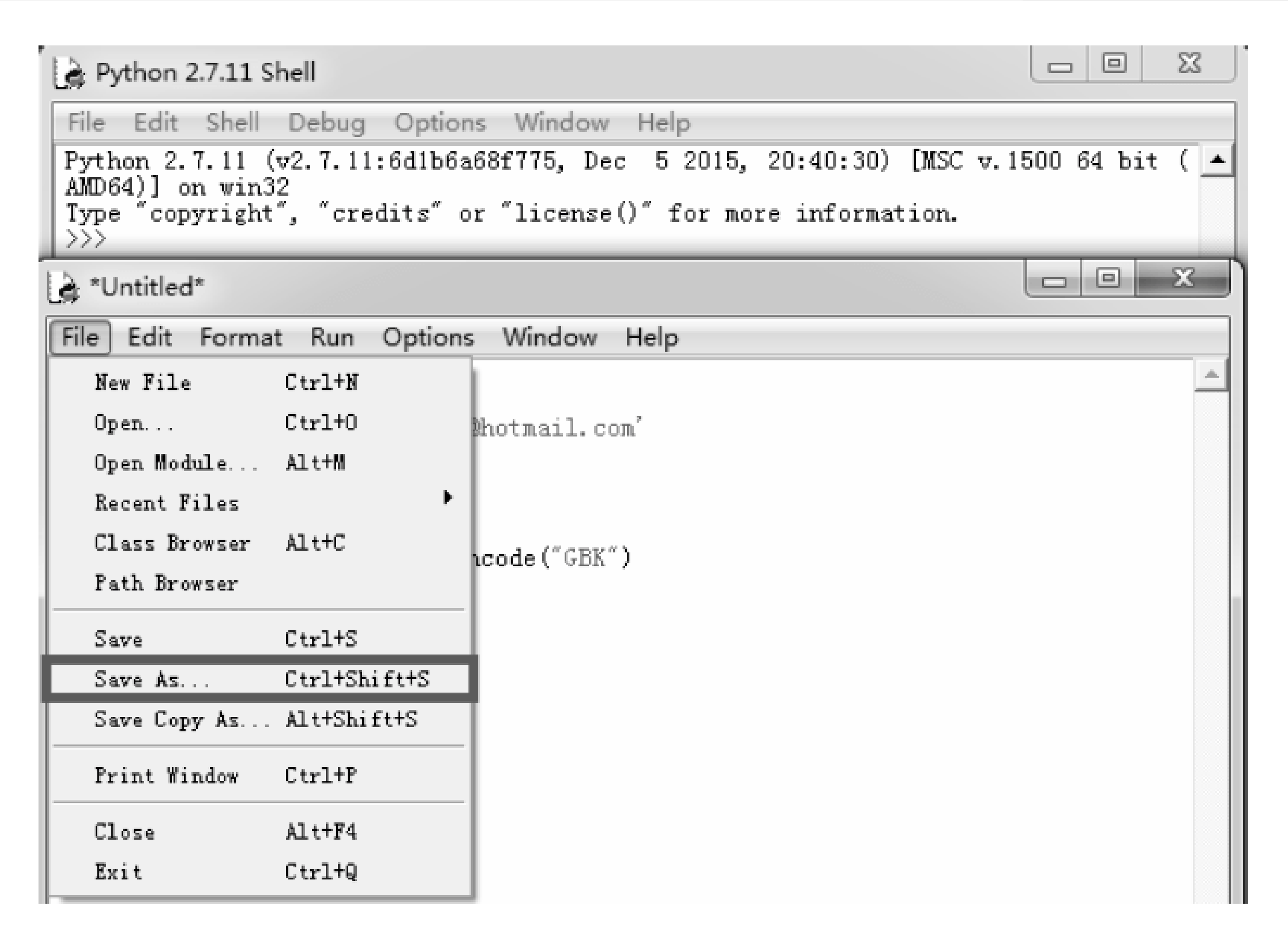


图 1-26　保存代码

（5）选择保存文件位置。这里选择的是保存到桌面，文件名为 hello.py，如图 1-27 所示。



图 1-27　选择文件保存位置

（6）单击“保存”按钮，将 hello.py 保存到桌面。按 Shift 按钮，同时右击桌面空白处，如图 1-28 所示。



图 1-28　打开 Windows 命令行工具

（7）单击“在此处打开命令窗口”，打开了命令行工具，执行命令：

```
python hello.py
```

执行结果如图 1-29 所示。

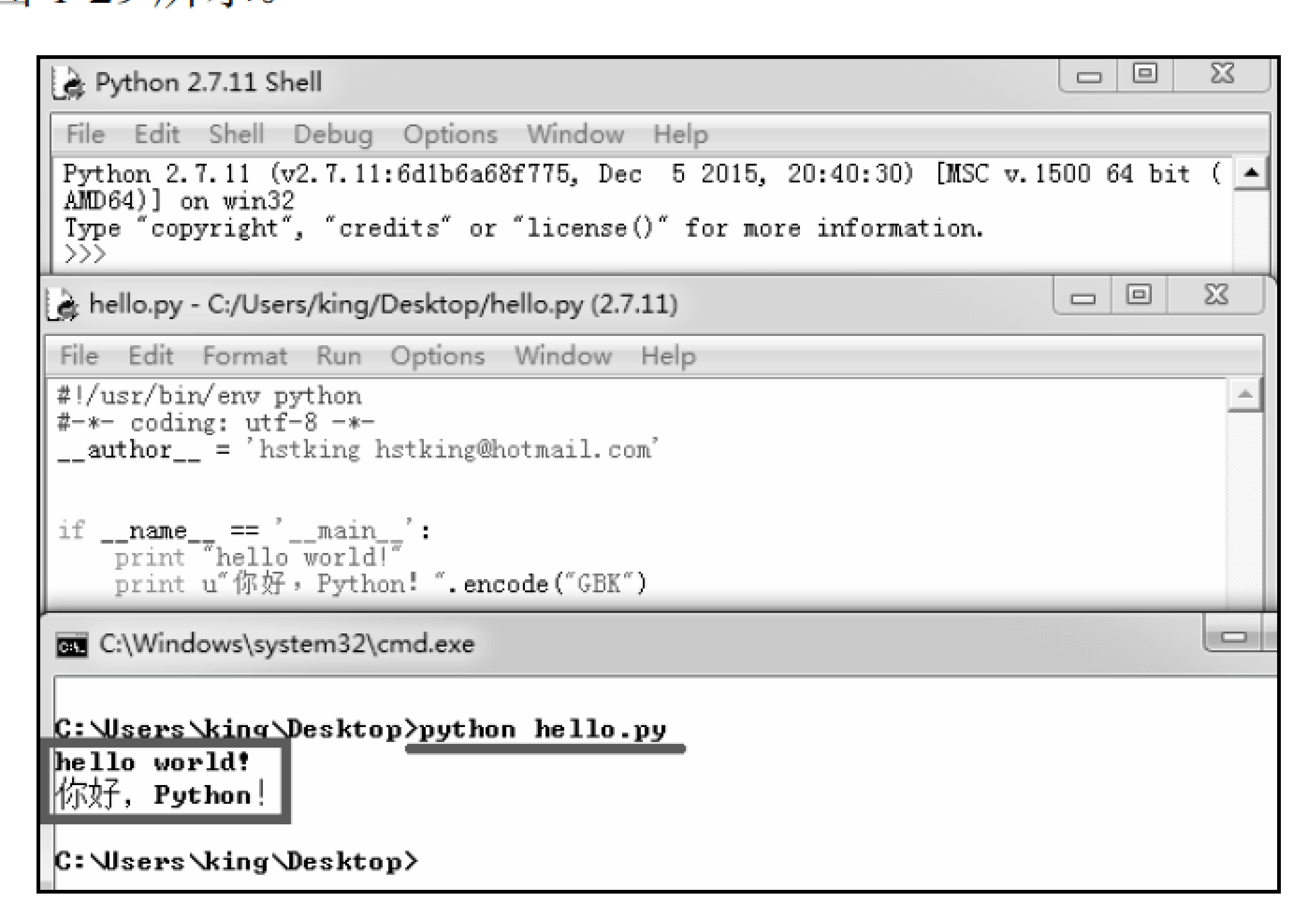


图 1-29　执行 hello.py

至此，Windows 下的 hello.py 执行完毕。

2．Linux 下创建 hello.py

（1）使用 Putty 连接到 Linux，执行命令：

```
mkdir -pv code/python
```

```
cd !$
cat > hello.py << EOF
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__authon__ = 'hstking hstking@hotmail.com'

if __name__ == '__main__':
    print("hello world!")
    print(u"你好，Python！ ")
EOF
```

执行结果如图 1-30 所示。

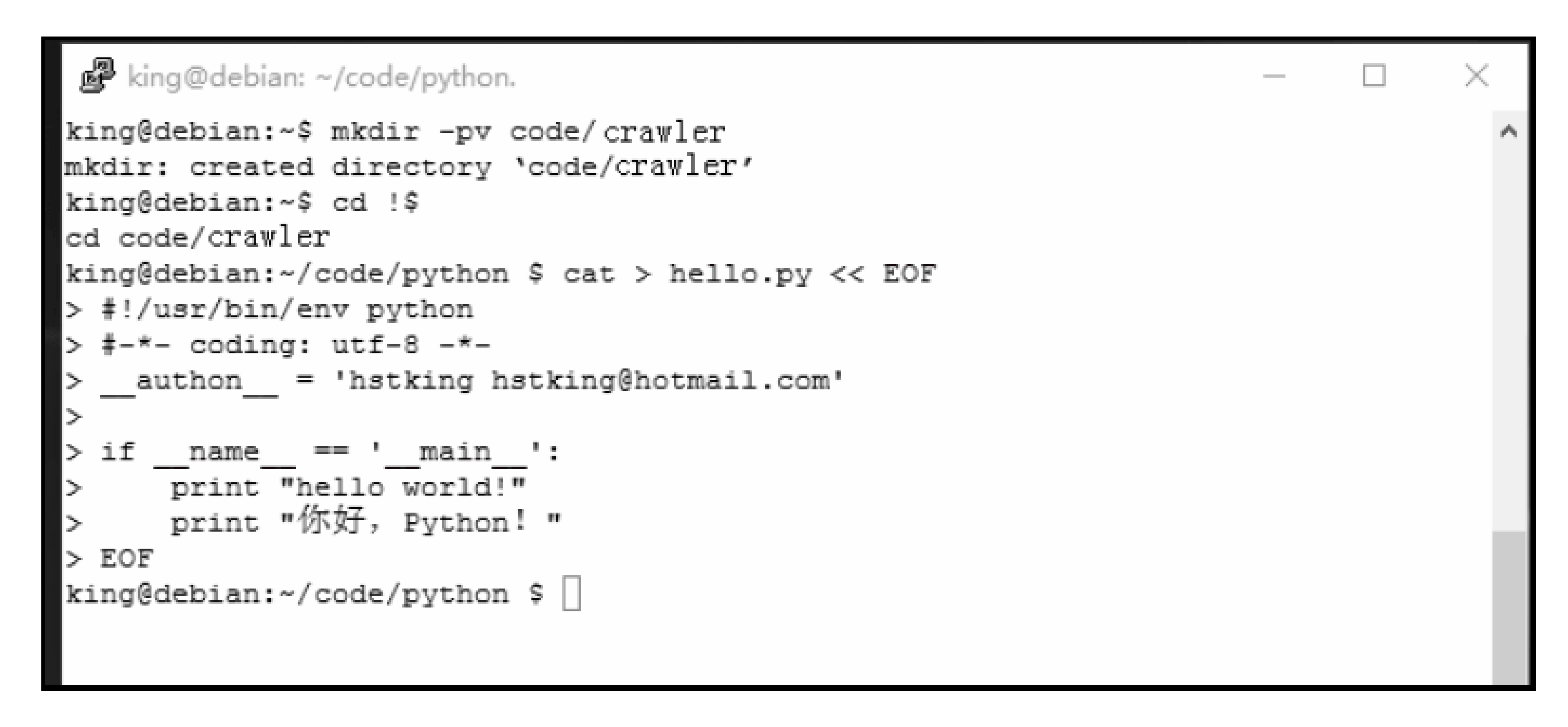


图 1-30　编辑 hello.py

（2）然后在 Putty 中执行命令：

```
python hello.py
```

执行结果如图 1-31 所示。

```
king@debian: ~/code/python.
king@debian:~$ mkdir -pv code/python.
mkdir: created directory 'code/python.'
king@debian:~$ cd !$
cd code/python.
king@debian:~/code/python.$ cat > hello.py << EOF
> #!/usr/bin/env python
> #-*- coding: utf-8 -*-
> __authon__ = 'hstking hstking@hotmail.com'
>
> if __name__ == '__main__':
>     print "hello world!"
>     print "你好，Python！ "
> EOF
king@debian:~/code/python.$ python hello.py
hello world!
你好，Python！
king@debian:~/code/python.$
```

图 1-31　执行 hello.py



这是没有使用文本编辑工具编辑文档，用的是 cat 命令。如果有条件，尽可能地使用文本编辑器，如 vi。几乎所有的 Linux 版本都默认安装了 vi 文本编辑器。

Linux 下的 hello.py 执行完毕。比较一下这两个不同系统下的 hello.py，可以发现基本上是一致的，只有在涉及中文处理的时候稍有差别（Windows 和 Linux 中的中文处理采用了不同的字符编码。Windows 使用的是 GBK，Linux 一般默认使用 utf8）。几乎所有的 Python 程序都可以在不同系统中通用，这也是 Python 最大的优势之一。

1.3 本章小结

Python 语言使用范围很广，能做最简单的数学加减运算，也能做高端的科学计算。既可服务于企业、政府、学校，也能用于个人，而且 Python 易学难精，不管是初学者还是“高段选手”都值得一学、一用。尤其是对网络的大力支持，使得 Python 用于网络编程具有很大的优势，这也是为什么要用 Python 写网络爬虫的原因之一。

第 2 章

◄ Python基础 ►

本章简略讲解 Python 的基础，介绍 Python 与其他编程语言的不同之处。在此主要是与 C 语言相比较。如果有 C 语言的基础，理解本章内容会更加容易；如果没有基础也没关系，Python 语言非常非常的简单，多看两遍也就会了。

2.1 Python 变量类型

Python 的标准数据类型只有 5 个，分别是数字、字符串、列表、元祖、字典。看起来比 C 语言的数据类型少了很多，但该有的功能一个不少。即使 C 语言的代表作链表和二叉树，Python 同样能应付自如。

2.1.1 数字

Python 支持 4 种不同的数值类型：

1. int 类型

有符号整数，就是 C 语言中所指的整型，也就是数学中的整数。它的大小与安装的解释器的位数有关。如果赋值超出了这个范围，则自动转换成了 Long 长整型。查看当前系统下的 Int 最大值，使用 Putty 登录 Linux，执行命令：

```
python
import sys
print sys.maxint
```

执行结果，如图 2-1 所示。

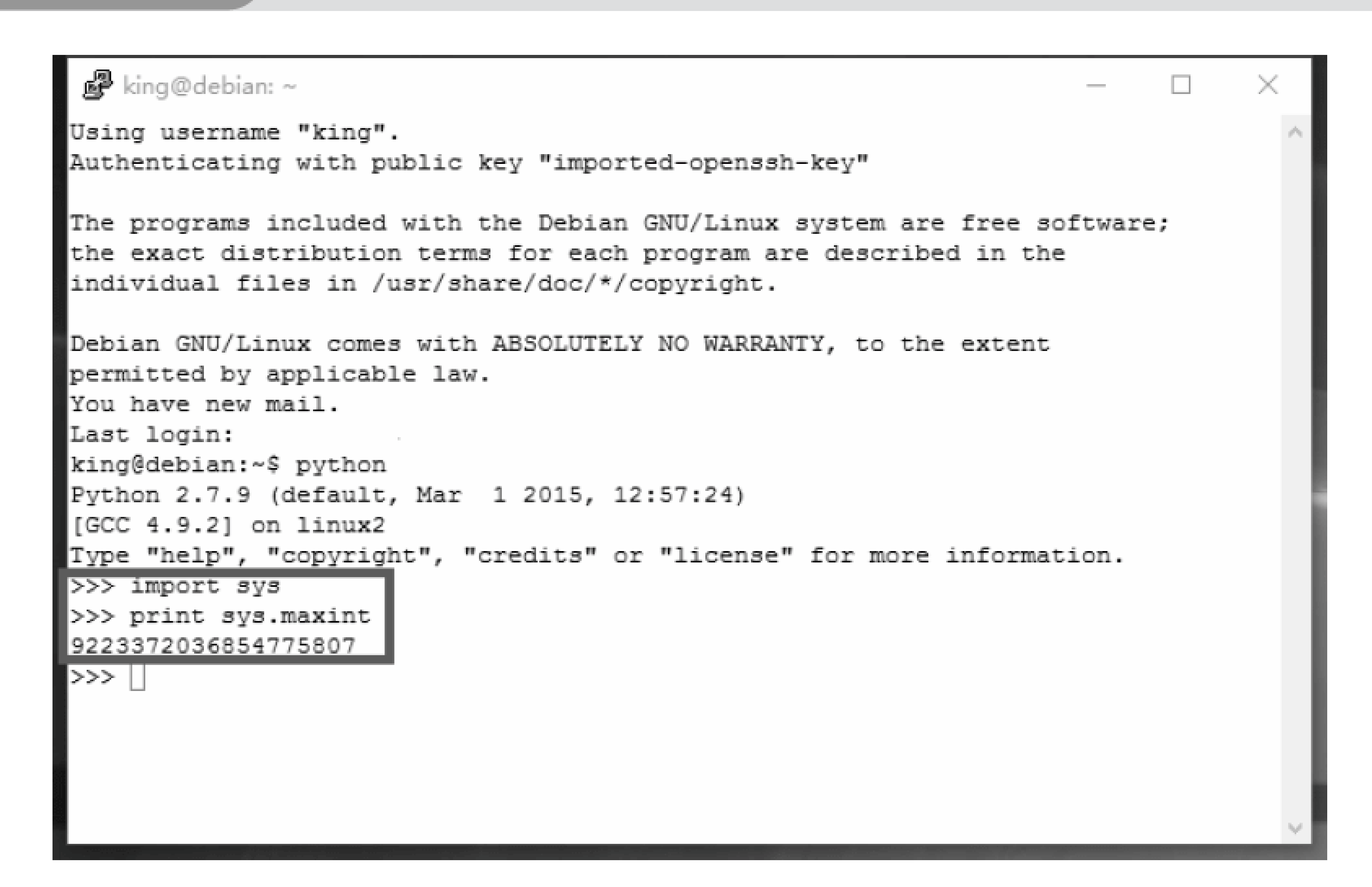


图 2-1　求 int 最大值

与 C 语言不同的是，Python 给变量赋值时不需要预先声明变量类型。也就是说，在给变量赋值时小于 9223372036854775807 的数字会被默认为 int 类型。超过了，则自动变成 Long 类型。

另外，八进制数字、十六进制数字都是属于 int（Long）类型的。

2. Long 类型

长整数，超过 int 类型的整数默认转换 Long，一般来说 int 足够用了。Long 类型没有限制大小，想赋值多大都行，只要内存足够大就可以了。在 Windows 中打开 cmd.exe，执行命令：

```
python
type(999999999999999999999999999999)
```

执行结果如图 2-2 所示。

```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\king>python
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500
AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> type(999999999999999999999999999999)
<type 'long'>
>>>
```

图 2-2　Python long int

3. Float 类型

浮点型实数，基本和 C 语言的浮点型一致，也就是数学中带小数点的数，不包括无限小数，不区分精度。只要是带小数点的数都可以看作浮点型数据。

4. Complex 类型

复数，在 C 语言中是需要自定义的一个数据类型。在 Python 中把它单独列出作为基本数据类型。复数包含一个有序对，表示为 a + bj，其中，a 是实部，b 是复数的虚部。

【示例 2-1】用一个简单的程序 showNumType.py 来显示 Python 的数字类型。使用 Putty 连接到 Linux，执行命令：

```
mkdir -pv code/crawler
cd !$
vi showNumType.py
```

showNumType.py 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

class ShowNumType(object):
    def __init__(self):
        self.showInt()
        self.showLong()
        self.showFloat()
        self.showComplex()

    def showInt(self):
        print(u"##########显示整型#############")
        print(u"十进制的整型")
        print("%-20d,%-20d,%-20d" %(-10000,0,10000))
        print(u"二进制的整型")
        print("%-20s,%-20s,%-20s" %(bin(-10000),bin(0),bin(10000)))
        print(u"八进制的整型")
        print("%-20s,%-20s,%-20s" %(oct(-10000),oct(0),oct(10000)))
        print(u"十六进制的整型")
        print("%-20s,%-20s,%-20s" %(hex(-10000),hex(0),hex(10000)))

    def showLong(self):
        print(u"##########显示长整型#############")
        print(u"十进制的整型")
        print("%-20Ld,%-20Ld,%-20Ld" %(-
1000000000000000000,0,1000000000000000000))
        print(u"八进制的整型")
        print("%-20s,%-20s,%-20s" %(oct(-
1000000000000000000),oct(0),oct(1000000000000000000)))
        print(u"十六进制的整型")
```

```
        print("%-20s,%-20s,%-20s" %(hex(-10000000000000000000),hex(0),hex(10000000000000000000)))

    def showFloat(self):
        print(u"##########显示浮点型#############")
        print("%-20.10f,%-20.10f,%-20.10f" %(-100.001,0,100.001))

    def showComplex(self):
        print(u"##########显示复数型#############")
        print(u"变量赋值复数 var = 3 + 4j")
        var = 3 + 4j
        print(u"var 的实部是：%d\tvar 的虚部是：%d" %(var.real,var.imag))


if __name__ == '__main__':
    showNum = ShowNumType()
```

在 Putty 下执行命令：

```
python showNumType.py
```

得到结果如图 2-3 所示。

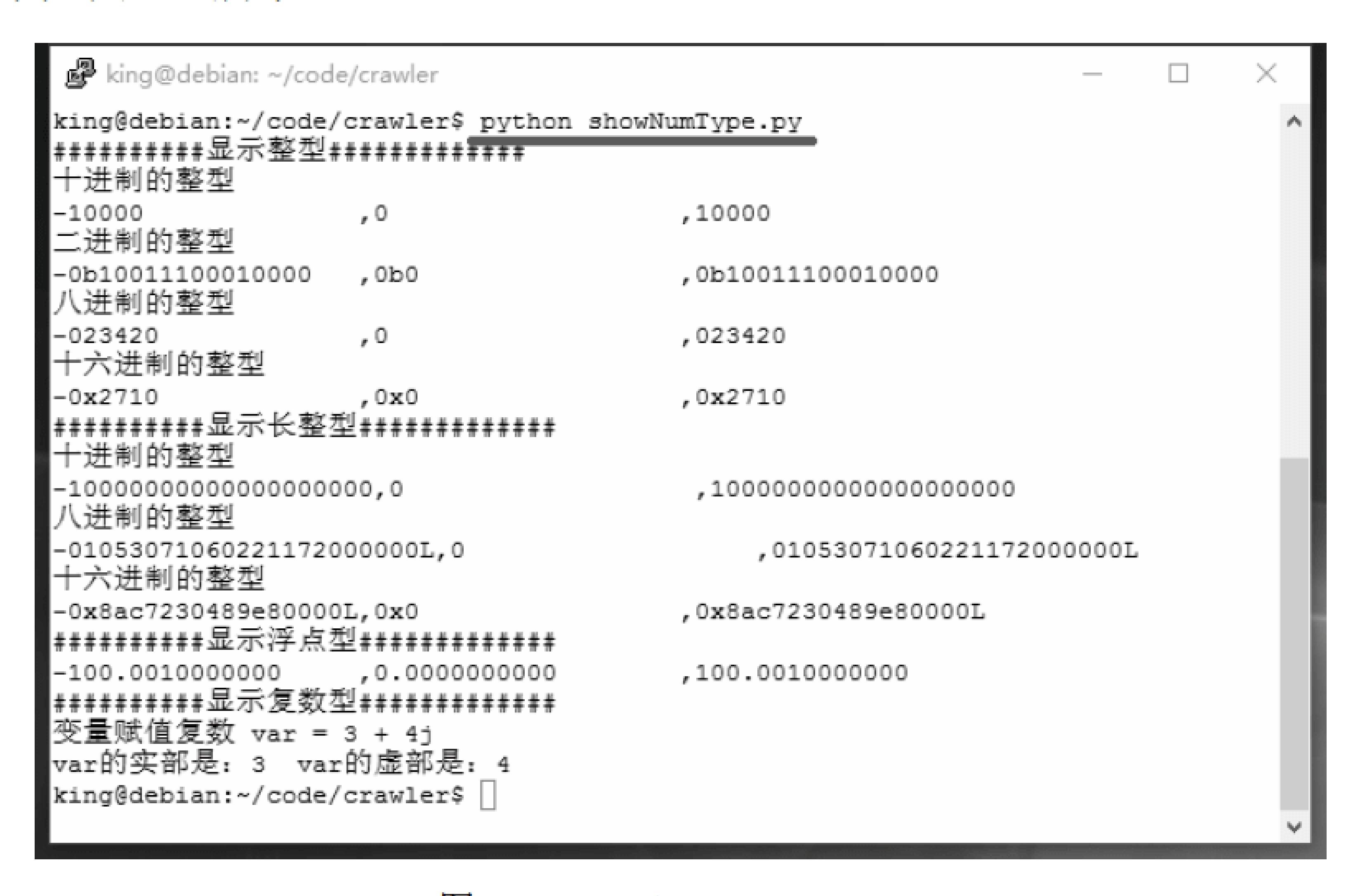


图 2-3 run showNumType.py

showNumType.py 是 Linux 下以 C++风格写的程序，展示如何标准输出各种基本数字类型。

2.1.2 字符串

在 Python 中，字符串是被定义为在引号（或双引号）之间的一组连续的字符。这个字符

可以是键盘上的所有可见字符，也可以是不可见的如“回车符”“制表符”等等。

字符串的操作方法很多，这里只选出最典型的几种。

（1）字符串大小写转换

- S.lower()：字母大写转换成小写。
- S.upper()：字母小写转换成大写。
- S.swapcase(): 字母大写转换小写，小写转换成大写。
- S.title(): 将首字母大写。

（2）字符串搜索、替换

- S.find(substr, [start, [end]]): 返回 S 中出现 substr 的第一个字母的标号，如果 S 中没有 substr 则返回-1，start 和 end 作用就相当于在 S[start:end]中搜索。
- S.count(substr, [start, [end]])：计算 substr 在 S 中出现的次数。
- S.replace(oldstr, newstr, [count]): 把 S 中的 oldstar 替换为 newstr，count 为替换次数。
- S.strip([chars]): 把 S 左右两端 chars 中有的字符全部去掉，一般用于去除空格。
- S.lstrip([chars]): 把 S 左端 chars 中所有的字符全部去掉。
- S.strip([chars]): 把 S 左端 chars 中所有的字符全部去掉。

（3）字符串分割、组合

- S.split([sep, [maxsplit]]): 以 sep 为分隔符，把 S 分成一个 list。maxsplit 表示分割的次数，默认的分割符为空白字符。
- S.join(seq): 把 seq 代表的序列——字符串序列，用 S 连接起来。

（4）字符串编码、解码

- S.decode([encoding]): 将以 encoding 编码的 S 解码成 unicode 编码。
- S.encode([encoding]): 将以 unicode 编码的 S 编码成 encoding，encoding 可以是 gb2312、gbk、big5……

（5）字符串测试

- S.isalpha(): S 是否全是字母，至少有一个字符。
- S.isdigit(): S 是否全是数字，至少有一个字符。
- S.isspace(): S 是否全是空白字符，至少有一个字符。
- S.islower(): S 中的字母是否全是小写。
- S.isupper(): S 中的字母是否全是大写。
- S.istitle(): S 是否是首字母大写的。

【示例 2-2】编写一个 showStrOperation.py 来实验一下。这次在 Windows 下以 IDLE 为 IDE 来编写程序。showStrOperation.py 代码如下：

```
#!/usr/bin/env python
#-*- coding: GBK -*-
'''因为 Windows 默认的中文字符编码是 GBK，所以…'''
__author__ = 'hstking hstking@hotmail.com'

def strCase():
    "字符串大小写转换"
    print "演示字符串大小写转换"
    print "演示字符串 S 赋值为：'  ThIs is a PYTHON  '"
    S = '  ThIs is a PYTHON  '
    print "大写转换成小写：\tS.lower() \t= %s"%(S.lower())
    print "小写转换成大写：\tS.upper() \t= %s"%(S.upper())
    print "大小写转换：\t\tS.swapcase() \t= %s"%(S.swapcase())
    print "首字母大写：\t\tS.title() \t= %s"%(S.title())
    print '\n'

def strFind():
    "字符串搜索、替换"
    print "演示字符串搜索、替换等"
    print "演示字符串 S 赋值为：'  ThIs is a PYTHON  '"
    S = '  ThIs is a PYTHON  '
    print "字符串搜索：\t\tS.find('is') \t= %s"%(S.find('is'))
    print "字符串统计：\t\tS.count('s') \t= %s"%(S.count('s'))
    print "字符串替换：\t\tS.replace('Is','is') = %s"%(S.replace('Is','is'))
    print "去左右空格：\t\tS.strip() \t=#%s#"%(S.strip())
    print "去左边空格：\t\tS.lstrip() \t=#%s#"%(S.lstrip())
    print "去右边空格：\t\tS.rstrip() \t=#%s#"%(S.rstrip())
    print '\n'

def strSplit():
    "字符串分割、组合"
    print "演示字符串分割、组合"
    print "演示字符串 S 赋值为：'  ThIs is a PYTHON  '"
    S = '  ThIs is a PYTHON  '
    print "字符串分割：\t\tS.split() \t= %s"%(S.split())
    print "字符串组合 1：  '#'.join(['this','is','a','python']) \t=
%s"%('#'.join(['this','is','a','python']))
    print "字符串组合 2：  '$'.join(['this','is','a','python']) \t=
%s"%('$'.join(['this','is','a','python']))
    print "字符串组合 3：  ' '.join(['this','is','a','python']) \t= %s"%('
'.join(['this','is','a','python']))
    print '\n'
```

```
    def strCode():
        "字符串编码、解码"
        print "演示字符串编码、解码"
        print "演示字符串 S 赋值为：'编码解码测试'"
        S = '编码解码测试'
        print "GBK 编码的 S \t = %s"%(S)
        print "GBK 编码的 S 转换 unicode 编码"
        print "S.decode('GBK')= %s"%(S.decode("GBK"))
        print "GBK 编码的 S 转换成 utf8"
        print "S.decode('GBK').encode('utf8') =
%s"%(S.decode("GBK").encode("utf8"))
        print "注意：不管是编码还是解码针对的都是 unicode 字符编码，\n 所以要编码或者解码前
必须先将源字符串转换成 unicode 编码格式"
        print '\n'

    def strTest():
        "字符串测试"
        print "演示字符串测试"
        print "演示字符串 S 赋值为：'abcd'"
        S1 = 'abcd'
        print "测试 S.isalpha() = %s"%(S1.isalpha())
        print "测试 S.isdigit() = %s"%(S1.isdigit())
        print "测试 S.isspace() = %s"%(S1.isspace())
        print "测试 S.islower() = %s"%(S1.islower())
        print "测试 S.isupper() = %s"%(S1.isupper())
        print "测试 S.istitle() = %s"%(S1.istitle())


    if __name__ == '__main__':
            strCase()
            strFind()
            strSplit()
            strCode()
            strTest()
```

打开 Windows 的命令行工具（cmd.exe），执行命令：

```
python showStrOperation.py
```

得到结果如图 2-4 所示。

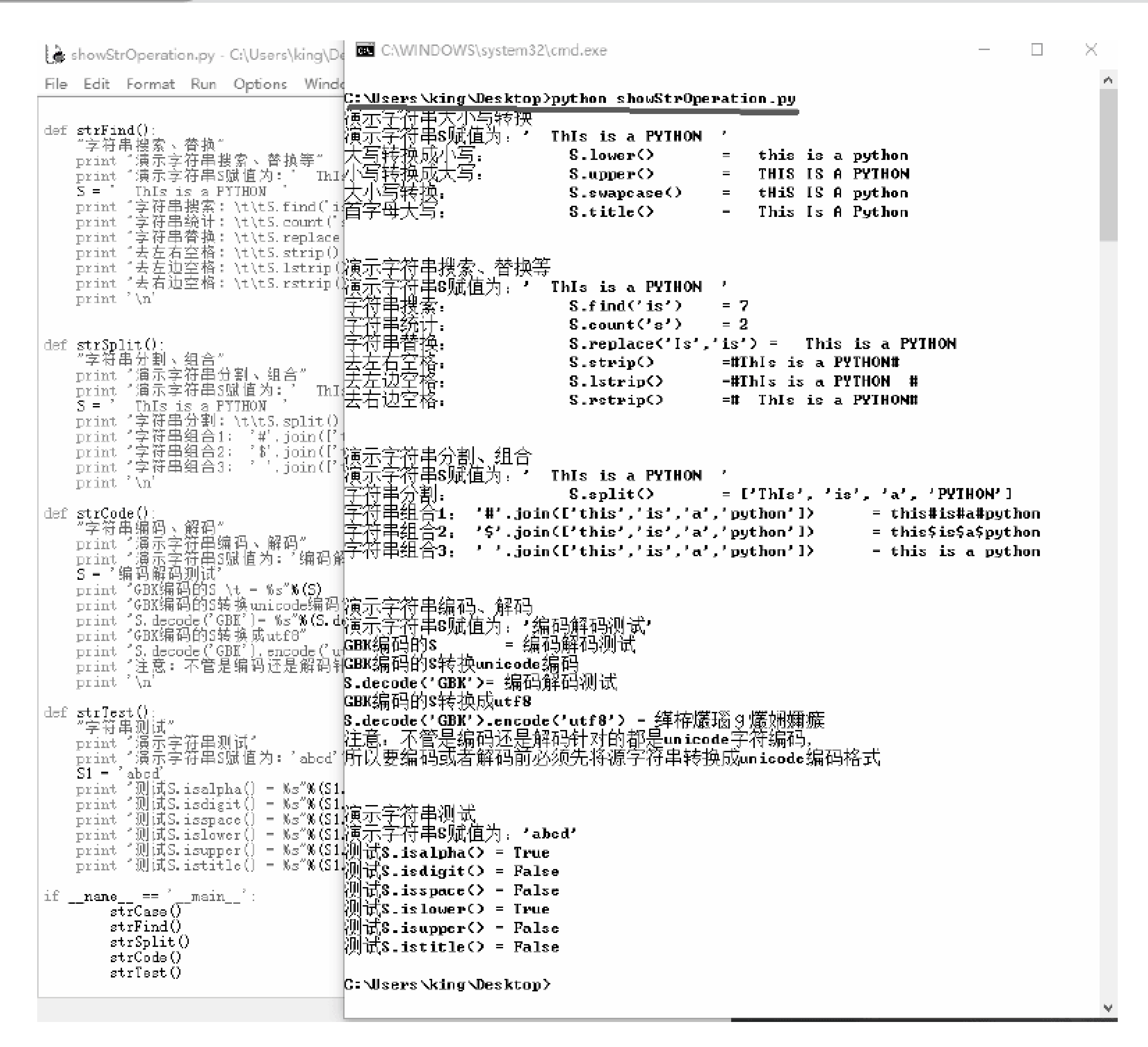


图 2-4　run showStrOperation.py

与 showNumType.py 不同，showStrOperation.py 是在 Windows 下以 C 语言的风格编写的。实际上这两个程序并没有什么区别，写哪种风格看个人习惯。唯一的区别就是 Windows 默认的是 GBK 编码，所以 showStrOperation.py 声明的是#-*- coding:GBK -*-；而 Linux 默认的是 utf8 编码，所以 showNumType.py 声明的是#-*- coding:utf8 -*-。



字符串也可以看成一个不可修改的字符列表。所以，大部分用来操作列表的方法（不涉及修改列表元素的）同样可以用来操作字符串。

2.1.3　列表

列表是 Python 最常用的变量类型。列表是一个可变序列，序列中的每个元素都分配一个数字即它的位置，或者叫索引。第一个索引是 0，第二个索引是 1，依此类推。列表中的元素可以是数字、字符串、列表、元组、字典……Python 使用中括号 [ ] 来解析列表，给一个变量赋值为空列表。很简单，执行命令 var = []就可以了。

列表的基本操作很简单，常用的操作一般是创建列表、插入数据、追加数据、访问数据、删除数据。下面实验一下：

创建列表，直接赋值即可。访问列表只需要列表名和列表中元素的下标即可。创建一个字符的列表，执行命令：

```
L1 = ['a','b','c','d','e']
```

```
L1[0]
L1[1]
L1[4]
L1[5]
```

执行结构如图 2-5 所示。

```
king@debian: ~
king@debian:~$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> L1 = ['a','b','c','d','e']
>>> L1
['a', 'b', 'c', 'd', 'e']
>>> L1[0]
'a'
>>> L1[1]
'b'
>>> L1[4]
'e'
>>> L1[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

图 2-5　创建列表

如图 2-5 所示，如果访问超出范围，Python 则会抛出一个异常 IndexError。如果只是创建一个纯字符的列表，无须一个个地输入字符，有个更简单的方法，执行命令 L1 = list('abcde')即可。

插入、追加、删除列表数据也很简单。执行命令：

```
L1.insert(0,0)
L1.insert(-1,100)
L1.append('python')
L1.pop(3)
L1.pop()
```

执行结果如图 2-6 所示。

```
king@debian: ~
>>> L1
['a', 'b', 'c', 'd', 'e']
>>> L1.insert(0,0)
>>> L1
[0, 'a', 'b', 'c', 'd', 'e']
>>> L1.insert(-1,100)
>>> L1
[0, 'a', 'b', 'c', 'd', 100, 'e']
>>> L1.append('python')
>>> L1
[0, 'a', 'b', 'c', 'd', 100, 'e', 'python']
>>> L1.pop(3)
'c'
>>> L1
[0, 'a', 'b', 'd', 100, 'e', 'python']
>>> L1.pop()
'python'
>>> L1
[0, 'a', 'b', 'd', 100, 'e']
>>>
```

图 2-6　插入、追加、删除数据

对列表最常用的操作是列表分片。分片可以简单地理解为将一个列表分成几块。它的操作方法是 list[index1:index2[:step]]。先创建一个较长的数字列表做这个分片示例，执行命令：

```
L2 = []
for i in xrange(0,101):
 L2.append(i)
L2
```

这样就创建了一个包含了从 0~100，共 101 个数字的列表，如图 2-7 所示。

```
king@debian: ~
>>> L2 = []
>>> for i in xrange(0,101):
...     L2.append(i)
...
>>> L2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 4
2, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 6
2, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 8
2, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
>>>
>>>
```

图 2-7 创建数字列表

列表切片其实和访问列表元素很相似。例如，要访问列表 L2 的第 10 个元素，那么就应该是 L2[10]就可以了。如果要访问列表 L2 的第 10 到 20 个元素呢？很简单，L2[10:21]就可以了。至于 list[index1:index2[:step]]中的 step 是步长。实验一下就清楚了，执行命令：

```
L2[0:21]
L2[21:41]
L2[81:100]
L2[0:21:1]
L2[0:21:2]
L2[0:21:3]
L2[0:21:4]
L2[0:21:5]
```

执行结果如图 2-8 所示。

```
king@debian: ~
>>> L2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 4
2, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 6
2, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 8
2, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
>>> L2[0:21]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
>>> L2[21:41]
[21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]
>>> L2[81:101]
[81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100
]
>>> L2[0:21:1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
>>> L2[0:21:2]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
>>> L2[0:21:3]
[0, 3, 6, 9, 12, 15, 18]
>>> L2[0:21:4]
[0, 4, 8, 12, 16, 20]
>>> L2[0:21:5]
[0, 5, 10, 15, 20]
>>>
```

图 2-8 列表分片

【示例 2-3】写个简单的程序 showList.py 验证一下。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi showList.py
```

showList.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

class ShowList(object):
    def __init__(self):
        self.L1 = []
        self.L2 = []

        self.createList()  #创建列表
        self.insertData()  #插入数据
        self.appendData()  #追加数据
        self.deleteData()  #删除数据
        self.subList()  #列表分片

    def createList(self):
        print("创建列表：")
        print("L1 = list('abcdefg')")
        self.L1 = list('abcdefg')
        print("L2 = []")
        print("for i in xrange(0,10):")
        print("\tL2.append(i)")
        for i in xrange(0,10):
            self.L2.append(i)
        print("L1 = "),
        print(self.L1)
        print("L2 = "),
        print(self.L2)
        print('\n')

    def insertData(self):
        print(u"插入数据")
        print(u"L1 列表中第 3 个位置插入数字 100，执行命令：L1.insert(3,100)")
        self.L1.insert(3,100)
        print("L1 = "),
        print(self.L1)
```

```
        print(u"L2 列表中第 10 个位置插入字符串'python', 执行命令:
L2.insert(10,'python')")
        self.L2.insert(10,'python')
        print("L2 = "),
        print(self.L2)
        print('\n')

    def appendData(self):
        print(u"追加数据")
        print(u"L1 列表尾追加一个列表[1,2,3], 执行命令 L1.append([1,2,3]")
        self.L1.append([1,2,3])
        print("L1 = "),
        print(self.L1)
        print(u"L2 列表尾追加一个元组('a','b','c'), 执行命令
L1.append(('a','b','c')")
        self.L2.append(('a','b','c'))
        print("L2 = "),
        print(self.L2)
        print('\n')

    def deleteData(self):
        print(u"删除数据")
        print(u"删除 L1 的最后一个元素, 执行命令 L1.pop()")
        self.L1.pop()
        print("L1 = "),
        print(self.L1)
        print(u"删除 L1 的第 1 个元素, 执行命令 L1.pop(0)")
        self.L1.pop(0)
        print("L1 = "),
        print(self.L1)
        print(u"删除 L2 的第 4 个元素, 执行命令 L1.pop(3)")
        self.L2.pop(3)
        print("L2 = "),
        print(self.L2)
        print('\n')

    def subList(self):
        print(u"列表分片")
        print(u"取列表 L1 的第 3 到最后一个元素组成的新列表, 执行命令 L1[2:]")
        print(self.L1[2:])
        print(u"取列表 L2 的第 2 个到倒数第 2 个元素组成的新列表, 步长为 2, 执行命>
令 L2[1:-1:2]")
        print(self.L2[1:-1:2])
```

```
        print('\n')


if __name__ == '__main__':
    print("演示列表操作：\n")
    sl = ShowList()
```

输入:wq，保存 showList.py。showList.py 显示了 Python 列表的基本功能——列表的创建、插入、追加、分片等。执行命令：

```
python showList.py
```

得到的结果如图 2-9 所示。

```
king@debian: ~/code/crawler
king@debian:~/code/crawler$ python showList.py
演示列表操作：

创建列表：
L1 = list('abcdefg')
L2 = []
for i in xrange(0,10):
        L2.append(i)
L1 =  ['a', 'b', 'c', 'd', 'e', 'f', 'g']
L2 =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


插入数据
L1列表中第3个位置插入数字100，执行命令：L1.insert(3,100)
L1 =  ['a', 'b', 'c', 100, 'd', 'e', 'f', 'g']
L2列表中第10个位置插入字符串'python'，执行命令：L2.insert(10,'python')
L2 =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'python']


追加数据
L1列表尾追加一个列表[1,2,3]，执行命令L1.append([1,2,3]
L1 =  ['a', 'b', 'c', 100, 'd', 'e', 'f', 'g', [1, 2, 3]]
L2列表尾追加一个元组('a','b','c')，执行命令L1.append(('a','b','c')
L2 =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'python', ('a', 'b', 'c')]


删除数据
删除L1的最后一个元素，执行命令L1.pop()
L1 =  ['a', 'b', 'c', 100, 'd', 'e', 'f', 'g']
删除L1的第1个元素，执行命令L1.pop(0)
L1 =  ['b', 'c', 100, 'd', 'e', 'f', 'g']
删除L2的第4个元素，执行命令L1.pop(3)
L2 =  [0, 1, 2, 4, 5, 6, 7, 8, 9, 'python', ('a', 'b', 'c')]


列表分片
取列表L1的第3到最后一个元素组成的新列表，执行命令L1[2:]
[100, 'd', 'e', 'f', 'g']
取列表L2的第2个到倒数第2个元素组成的新列表，步长为2，执行命令L2[1:-1:2]
[1, 4, 6, 8, 'python']
Traceback (most recent call last):
  File "showList.py", line 82, in <module>
    sl = ShowList()
  File "showList.py", line 15, in __init__
    self.subList()
  File "showList.py", line 78, in subList
    pritn('\n')
NameError: global name 'pritn' is not defined
king@debian:~/code/crawler$
```

图 2-9　run showList.py

列表还有很多其他的函数和操作方法，如有兴趣可参考官方文档和 google。列表和元组非常相似，掌握了列表，就基本掌握了元组。列表，是 Python 编程中必不可少的一部分。

2.1.4 元组

Python 的元组与列表非常相似，不同之处在于元组的元素是不可修改，它是一个不可变序列。列表使用[]来声明，元组使用()声明。

元组创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。创建一个空元组，执行命令 var = ()。因为元组中元素是不可修改的，所以列表中的操作方法 insert、append、pop 等操作对于元组这些都没有。又因为元组与列表的高度相似性，列表的切片对元组是完全适用的（切片并不改变原始数据），所以只需要记住一个原则，列表中修改元素值的操作元组都不可用，列表中不修改元素值的操作元组基本上都可以用。

元组和列表是可以互相转换的。使用 tuple(list)可以将一个列表转换成元组，反过来使用 list(tuple)也可以将一个元组转换成列表。

【示例 2-4】编写一个 showTuple 来实验一下。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi showTuple.py
```

showTuple.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

class ShowTuple(object):
    def __init__(self):
        self.T1 = ()
        self.createTuple()  #创建元组
        self.subTuple(self.T1)  #元组分片
        self.tuple2List(self.T1)  #元组、列表转换

    def createTuple(self):
        print(u"创建元组：")
        print(u"T1 = (1,2,3,4,5,6,7,8,9,10)")
        self.T1 = (1,2,3,4,5,6,7,8,9,10)
        print(u"T1 = "),
        print(self.T1)
        print('\n')

    def subTuple(self,Tuple):
        print(u"元组分片：")
        print(u"取元组 T1 的第 4 个到最后一个元组组成的新元组，执行命令 T1[3:]")
        print(self.T1[3:])
        print(u"取元组 T1 的第 2 个到倒数第 2 个元素组成的新元组，步长为 2，执行命令 T1[1:-1:2]")
```

```
        print(self.T1[1:-1:2])
        print('\n')

    def tuple2List(self,Tuple):
        print(u"元组转换成列表：")
        print(u"显示元组")
        print(u"T1 = "),
        print(self.T1)
        print(u"执行命令 L2 = list(T1)")
        L2 = list(self.T1)
        print(u"显示列表")
        print(u"L2 = "),
        print(L2)
        print(u"列表追加一个元素 100 后，转换成元组。执行命令 L2.append(100) tuple(L2)")
        L2.append(100)
        print(u"显示新元组")
        print(tuple(L2))


if __name__ == '__main__':
    st = ShowTuple()
```

输入:wq，保存 showTuple.py。showTuple.py 显示了 Python 元组的创建、分片和转换。执行命令：

```
python showTuple.py
```

得到的结果如图 2-10 所示。

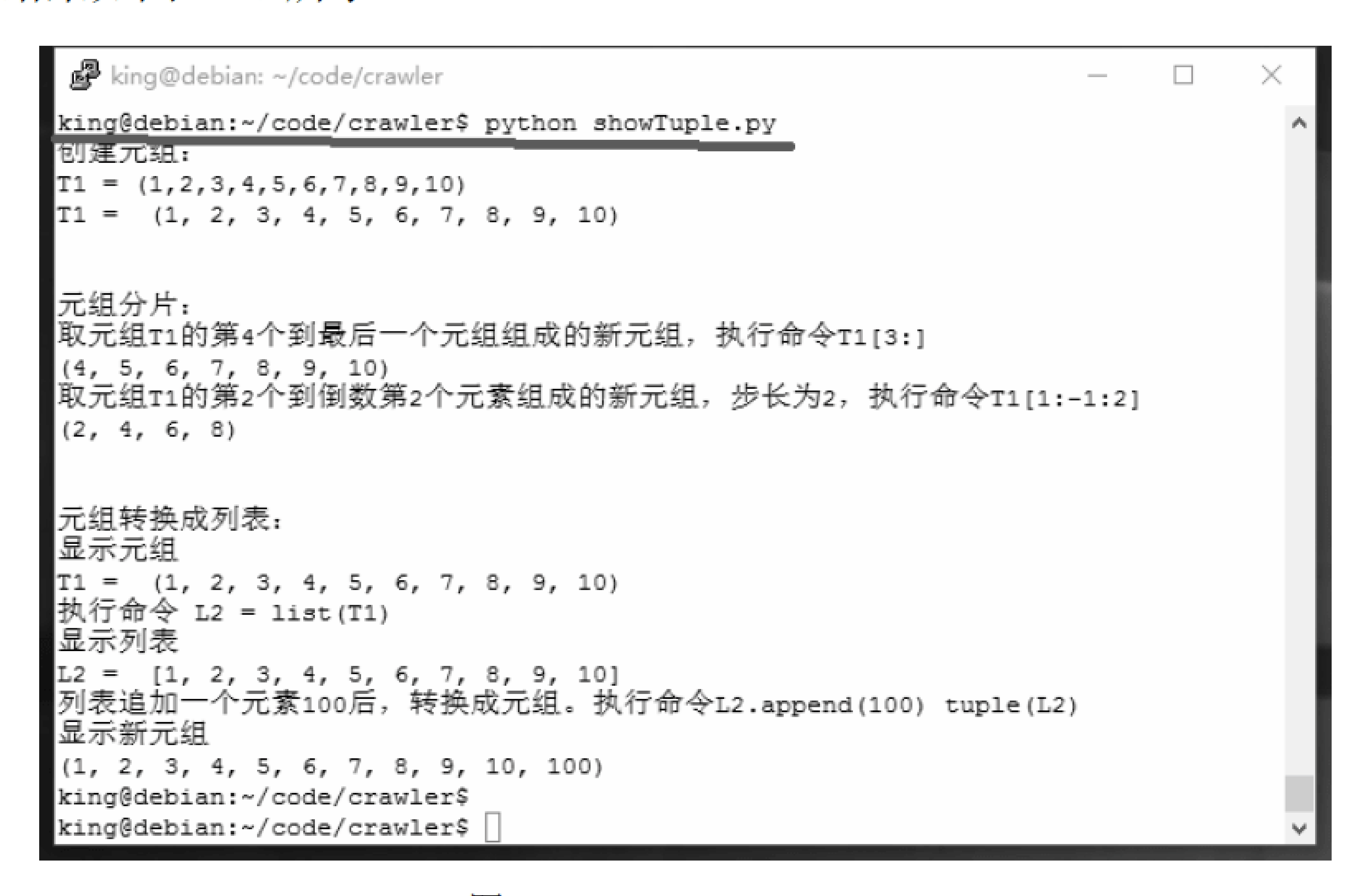


图 2-10　run showTuple.py

因为元组和列表高度相似，绝大部分场合都可以用列表来替代元组。



元组和列表不同仅在于一个可修改，一个不可修改。其他方面几乎没什么区别。由于元组不可修改的特性，一般在函数中需要返回多个返回值时，可以将这些返回值放入一个元组中返回。

2.1.5 字典

从某种意义上来说，字典和列表也很相似。字典使用的是{}，列表使用的是[]，元素分隔符都是逗号。所不同的是列表的索引只是从 0 开始的有序整数，不可重复；而字典的索引实际上在字典里应该叫键。虽然字典中的键和列表中的索引一样是不可重复的，但键是无序的，也就是说字典中的元素是没有顺序而言的。字典中的元素任意排列都不影响字典的使用。

字典的键可以是数字、字符串、列表、元组……几乎什么都可以，一般用字符串来做键，键与键值用冒号分割。在列表中是通过索引来访问元素，而在字典中是通过键来访问键值。因为字典按“键”寻值而不同于列表的按“索”寻值，所以字典的操作方法与列表稍有区别。

首先创建一个字典实验一下，执行命令：

```
ironMan = {'name':'tony stark','age':47,'sex':'male'}
```

这样就建立了一个简单的 IronMan 字典。因为字典的键值是无序的，所以插入一个数据无须 insert 之类的方法。直接定义即可，执行命令：

```
ironMan['college'] = 'NYU'
ironMan['Nation'] = 'America'
```

如需添加资料，只需要继续这样添加即可。如果发现资料有误，修改字典，同样也是直接定义，执行命令：

```
ironMan['college'] = 'MIT'
```

如果要删除某个元素，可以使用 del 命令。del 命令可以理解为取消分配给变量的内存空间。执行命令：

```
del ironman['Nation']
```

del 命令不只是可以删除字典的元素，类似字典元素、用户定义的变量都可以用 del 来删除。它可以删除数字变量、字符串变量、列表、元组、字典等等。

字典还有一些独特的操作。以下是字典中最常用的操作：

- dcit.keys()：返回一个包含字典所有 key 的列表。
- dict.values()：返回一个包含字典所有 value 的列表。
- dict.items()：返回一个包含所有(键,值)元组的列表。

- dict.clear()：删除字典中所有的元素。
- dict.get(key)：返回字典中 key 所对应的值。

【示例 2-5】编写一个 showDict 来实验一下。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi showDict.py
```

showDict.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

class ShowDict(object):
    '''该类用于展示字典的使用方法 '''
    def __init__(self):
        self.spiderMan = self.createDict()  #创建字典
        self.insertDict(self.spiderMan)  #插入元素
        self.modifyDict(self.spiderMan)  #修改元素
        self.operationDict(self.spiderMan)  #字典操作
        self.deleteDict(self.spiderMan)  #删除元素

    def createDict(self):
        print(u"创建字典:")
        print(u"执行命令 spiderMan = {'name':'Peter Parker','sex':'male','Nation':'Americ','college':'MIT'}")
        spiderMan = {'name':'Peter Parker','sex':'male','Nation':'Americ','college':'MIT'}
        self.showDict(spiderMan)
        return spiderMan

    def showDict(self,spiderMan):
        print(u"显示字典")
        print(u"spiderMan = "),
        print(spiderMan)
        print('\n')

    def insertDict(self,spiderMan):
        print(u"字典中添加键 age，值为 31")
        print(u"执行命令 spiderMan['age'] = 31")
        spiderMan['age'] = 31
        self.showDict(spiderMan)

    def modifyDict(self,spiderMan):
```

```
        print(u"字典修改键'college'的值为'Empire State University'")
        print(u"执行命令 spiderMan['college'] = 'Empire State University'")
        spiderMan['college'] = 'Empire State University'
        self.showDict(spiderMan)

    def operationDict(self,spiderMan):
        print(u"字典的其他操作方法")
        print(u"###########################")
        print(u"显示字典所有的键，keyList = spiderMan.keys()")
        keyList = spiderMan.keys()
        print(u"keyList = "),
        print(keyList)
        print('\n')
        print(u"显示字典所有键的值，valueList = spiderMan.values()")
        valueList = spiderMan.values()
        print(u"valueList = "),
        print(valueList)
        print('\n')
        print(u"显示字典所有键和值的元组，itemList = spiderMan.items()")
        itemList = spiderMan.items()
        print(u"itemList = "),
        print(itemList)
        print('\n')
        print(u"取字典中键为 college 的值,college = spiderman.get('college')")
        college = spiderMan.get('college')
        print(u"college = %s" %college)
        print('\n')

    def deleteDict(self,spiderMan):
        print(u"删除字典中键为 Nation 的值")
        print(u"执行命令 del(spiderMan['Nation'])")
        del(self.spiderMan['Nation'])
        self.showDict(spiderMan)
        print(u"清空字典中所有的值")
        print(u"执行命令 spiderMan.clear()")
        self.spiderMan.clear()
        self.showDict(spiderMan)
        print(u"删除字典")
        print(u"执行命令 del(spiderMan)")
        del(spiderMan)
        print(u"显示 spiderMan")
        try:
```

```
            self.showDict(spiderMan)
        except NameError:
            print(u"spiderMan 未被定义")


if __name__ == '__main__':
    sd = ShowDict()
```

输入:wq，保存 showDict.py。showDict.py 显示了 Python 字典的基本功能。执行命令：

```
python showDict.py
```

得到的结果如图 2-11 所示。

```
king@debian: ~/code/crawler
king@debian:~/code/crawler$ python showDict.py
创建字典:
执行命令spiderMan = {'name':'Peter Parker','sex':'male','Nation':'Americ','colle
ge':'MIT'}
显示字典
spiderMan =  {'Nation': 'Americ', 'college': 'MIT', 'name': 'Peter Parker', 'sex
': 'male'}


字典中添加键age, 值为31
执行命令spiderMan['age'] = 31
显示字典
spiderMan =  {'age': 31, 'Nation': 'Americ', 'college': 'MIT', 'name': 'Peter Pa
rker', 'sex': 'male'}


字典修改键'college'的值为'Empire State University'
执行命令spiderMan['college'] = 'Empire State University'
显示字典
spiderMan =  {'age': 31, 'Nation': 'Americ', 'college': 'Empire State University
', 'name': 'Peter Parker', 'sex': 'male'}


字典的其它操作方法
###########################
显示字典所有的键, keyList = spiderMan.keys()
keyList =  ['age', 'Nation', 'college', 'name', 'sex']


显示字典所有键的值, valueList = spiderMan.values()
valueList =  [31, 'Americ', 'Empire State University', 'Peter Parker', 'male']


显示字典所有键和值的元组, itemList = spiderMan.items()
itemList =  [('age', 31), ('Nation', 'Americ'), ('college', 'Empire State Univer
sity'), ('name', 'Peter Parker'), ('sex', 'male')]


取字典中键为college的值,college = spiderman.get('college')
college = Empire State University


删除字典中键为Nation的值
执行命令 del(spiderMan['Nation'])
显示字典
spiderMan =  {'age': 31, 'college': 'Empire State University', 'name': 'Peter Pa
rker', 'sex': 'male'}


清空字典中所有的值
执行命令 spiderMan.clear()
显示字典
spiderMan =  {}


删除字典
执行命令 del(spiderMan)
显示spiderMan
spiderMan 未被定义
king@debian:~/code/crawler$
```

图 2-11　run showDict.py

Python 的基本变量类型就是这些。其他的类型几乎都是由这些基本类型组合而来（Python 的数据类型还有 None 和 boolean）。



字典的键和键值可以是任何类型。在没有什么特殊要求的情况下尽可能地使用字符串作为键。如果把键设置得太复杂了，那也就失去字典的意义了。

2.2 Python 语句

说到语句，回想一下 C、C++、Java、Perl 等，似乎所有的编程语言都有类似的语句。条件判断、有限循环、无限循环，这几个是最基本的，也是必不可少的。每个编程语言都差不多。熟悉了这几个语句后，即使是一门从未接触过的语言，稍微了解一下格式语法就可以用新的语言解决一般的小问题了。

2.2.1 条件语句——if else

似乎所有的条件语句都使用 if……else……。它的作用可以简单地概括为非此即彼。满足条件 A 则执行 A 的语句，否则执行 B 语句。Python 的 if.……else……功能更加强大，在 if 和 else 之间添加数个 elif，有更多的条件选择。其表达形式如下：

```
if 判断条件 1:
 执行语句 1
elif 判断条件 2:
 执行语句 2
elif 判断条件 3:
 执行语句 3
else:
 执行语句 4
```

【示例 2-6】编写 testIfRemainder7.py 熟悉一下 Python 下的 if 语句。testIfRemainder7.py 用来检验输入数字能否被 7 整除。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testIfRemainder7.py
```

testIfRemainder7.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


def isEvenNum(num):
```

```
    if num%7 == 0:
        print(u"%d 可以被 7 整除" %num)
    else:
        print(u"%d 不可被 7 整除" %num)

if __name__ == '__main__':
    numStr = raw_input("请输入一个整数：")
    try:
        num = int(numStr)
    except ValueError:
        print(u"输入错误，要求输入一个整数")
        exit()

    isEvenNum(num)
```

输入:wq，保存 testIfRemainder7.py。testIfRemainder7.py 要求用户输入一个整数，然后判断这个数能否被 7 整除，基本就是一个最基本的非此即彼的判断。执行命令：

```
python testIfRemainder7.py
```

得到的结果如图 2-12 所示。

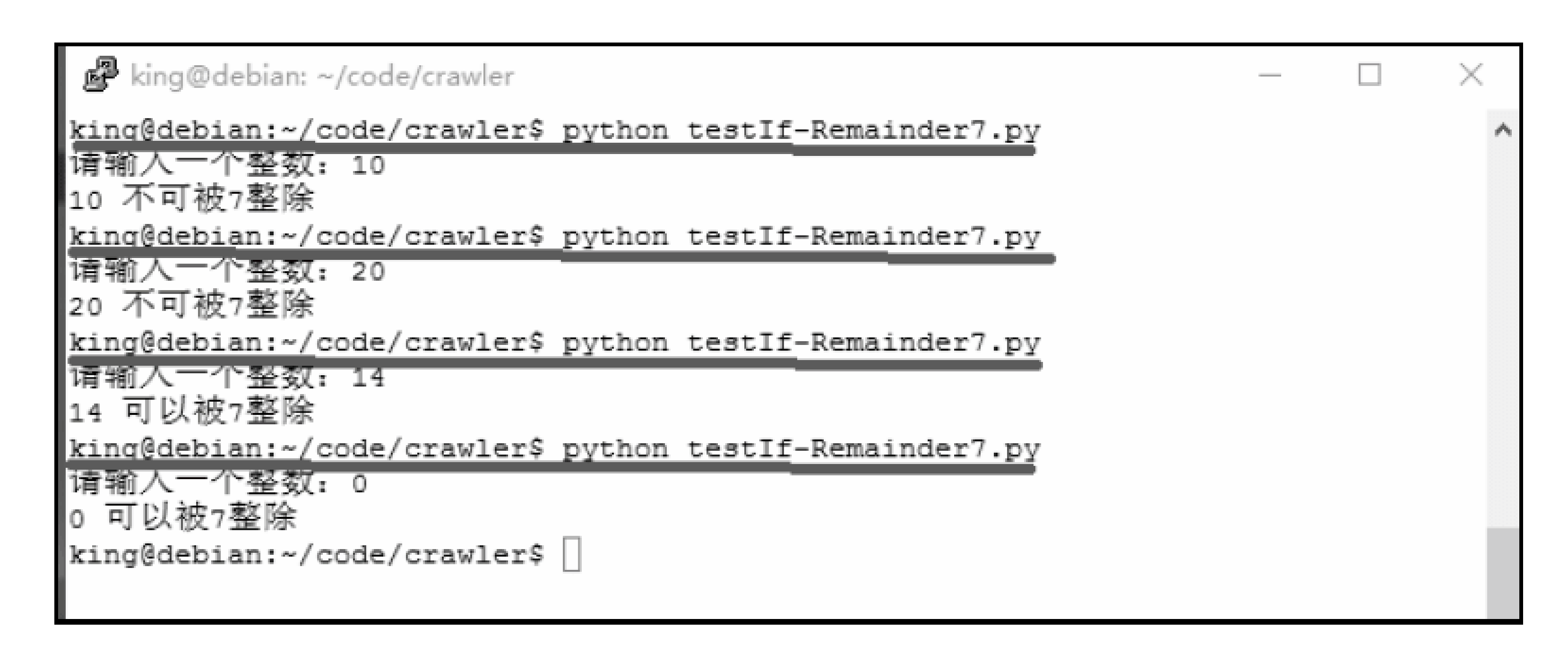


图 2-12　run testIfRemainder7.py

非常简单。按照格式，照猫画虎就可以解决类似的问题了。

Case switch 是 C 语言中经典的条件语句之一。可惜的是 Python 中并没有 Case 语句。不过没关系，if elif else 完全可以替代 case 语句。如果原意开动脑筋，Python 中还有很多可以替代 Case 语句的方案，例如利用字典什么的，这里就不再一一赘述了。

2.2.2　有限循环——for

在编程时，总会遇到这种事情，把某个过程重复 N 次。这是每个编程语言都不可避免的。好在几乎所有的编程语言都提供 for 语句。它的作用是将一个语句块、函数等重复执行有限的次数。

for 循环表达形式如下：

```
for Var in Sequence:
 执行语句
```

比如从 1 加到 100。大数学家高斯（Johann Karl Friedrich Gauss）10 岁时就给出了计算的公式。虽然已经有了简单的方法，用笨方法验算一下也不错。

【示例 2-7】编写 testForGauss10.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testForGauss10.py
```

testForGauss10.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

def cumulative(num):
    sum = 0
    for i in xrange(1,num+1):
        sum += i
    return sum  #累加函数，返回累加后的值

def main():
    while True:
        print(u"============================")
        print(u"输入 exit 退出程序:")
        str_num = raw_input("从 1 累加到: ")
        if str_num == 'exit':
            break
        try:
            sum = cumulative(int(str_num))
        except ValueError:
            print(u"除非退出输入 exit，只能输入数字")
            continue
        print(u"从 1 累加到%d 的总数是%d" %(int(str_num),sum))


if __name__ == '__main__':
    main()
```

输入:wq，保存 testForGauss10.py。testForGauss.py 将使用最笨的方法求从 1 加到 100 的和，使用 for 循环一个数一个数地叠加。执行命令：

```
python testForGauss10.py
```

得到的结果如图 2-13 所示。

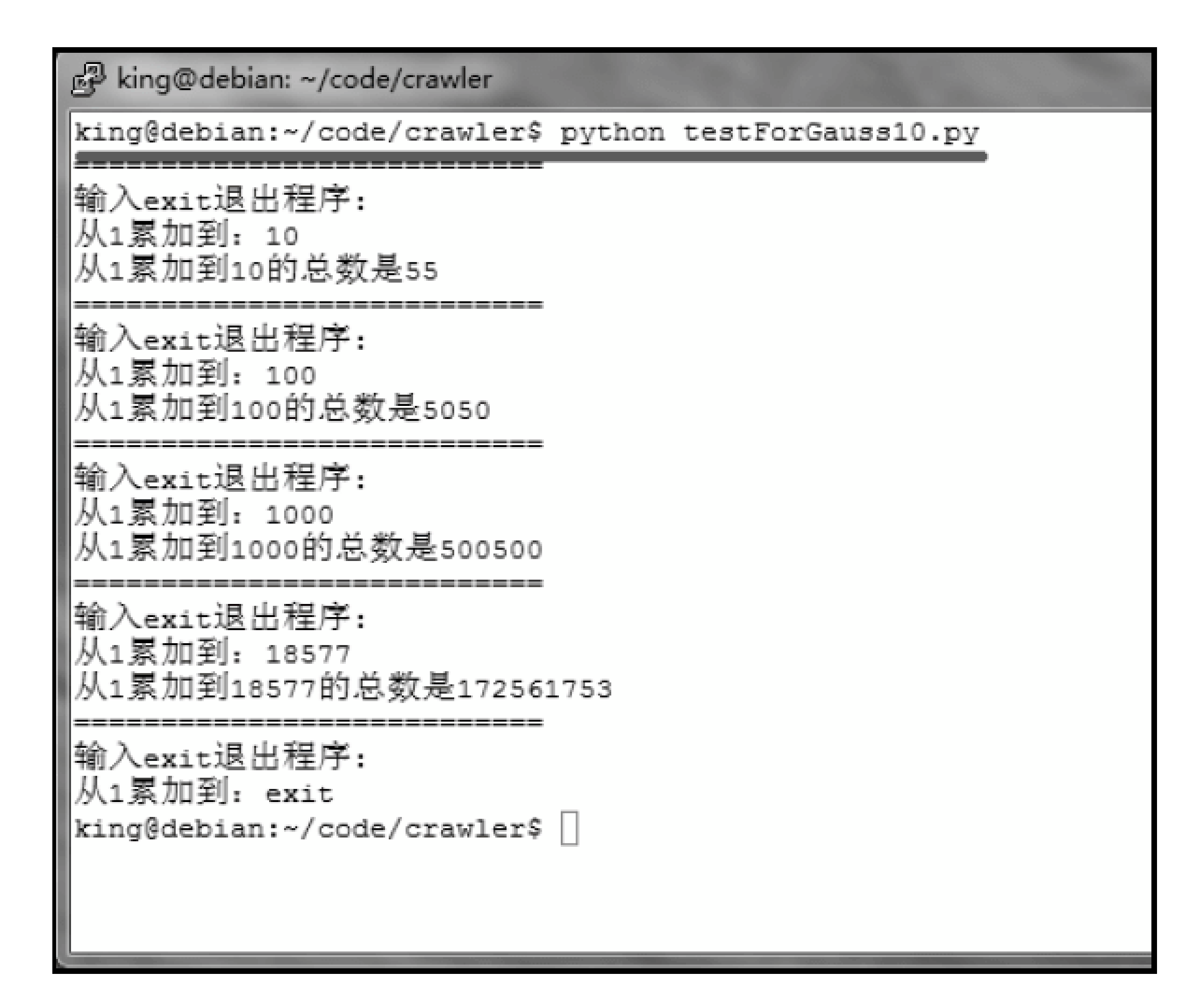


图 2-13　run testForGauss10.py

经过验算，聪明办法和笨办法得到的结果一致。for 循环用于数字循环时有 2 种方法生成 Sequence，一种是 range(1,100)，另一种是 xrange(1,100)。在使用 for 循环时，这两种方法生成的 Sequence 几乎没有区别。但如果循环数比较大的情况下建议使用 xrange，因为 range 是直接生成了一个列表，而 xrange 则是生成了一个生成器。

仔细看下 for 循环的表达式，Sequence 是一个序列，说明 for 循环不仅仅适用于数字形式的循环，比如可以将文件放入列表中作为一个序列，然后对文件进行操作。

2.2.3　无限循环——while

既然有有限循环，当然就有无限循环了。无限循环的作用是，只要不满足某种条件，就一直循环下去，直到满足条件为止。While 循环表达形式如下：

```
while Boolean expression:
 执行语句
```

【示例 2-8】Linux 终端登录就是一个类似 while 循环的示例。下面模拟 Linux 终端登录，编写 testWhileSimulateLogin.py。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testWhileSimulateLogin.py
```

testWhileSimulateLogin.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import getpass

```

```
class FakeLogin(object):
    def __init__(self):
        self.name = 'king'
        self.password = 'haha,no pw'
        self.banner = 'hello, you have login system'
        self.run()

    def run(self):
        '''仿 Linux 终端登录窗口'''
        print(u"不好意思，只有一个用户 king")
        print(u"偷偷的告诉你，密码是 6 个 8 哦")
        while True:
            print(u"Login:king")
            pw = getpass.getpass("Password:")
            if pw == '88888888':
                print(u"%s" %self.banner)
                print(u"退出程序")
                exit()
            else:
                if len(pw) > 12:
                    print(u"密码长度应该小于 12")
                    continue
                elif len(pw) < 6:
                    print(u"密码长度大于 6 才对")
                    continue
                else:
                    print(u"可惜，密码错误。继续猜")
                    continue


if __name__ == '__main__':
    fl = FakeLogin()
```

输入:wq，保存 testWhileSimulateLogin.py。testWhileSimulateLogin.py 脚本模拟 Linux 登录，如果输入了正确的密码才退出程序，输入了错误的密码则给出相应的提示，直到输入正确为止。因为不知道会输入多少次才会退出，所以这里使用 while 循环正好。执行命令：

```
python testWhileSimulateLogin.py
```

得到的结果如图 2-14 所示。

```
king@debian:~/code/crawler$ python testWhileSimulateLogin.py
不好意思，只有一个用户king
偷偷的告诉你，密码是6个8哦
Login:king
Password:
密码长度应该小于12
Login:king
Password:
密码长度大于6才对
Login:king
Password:
可惜，密码错误。继续猜
Login:king
Password:
hello, you have login system
退出程序
king@debian:~/code/crawler$
```

图 2-14　run testWhileSimulateLogin.py

实际上目前的终端登录都有次数限制，不可能这样无限地输入密码进行测试，否则就会被暴力破解。正好这个程序没有限制，有兴趣的可以自行编写程序，实验一下暴力破解密码。

2.2.4　中断循环——continue、break

continue 和 break 语句都只能作用于循环之中，只对循环起作用。continue 的作用是，从 continue 语句开始到循环结束，之间所有的语句都不执行，直接从下一次循环重新开始；而 break 语句的作用是退出循环，该循环结束。

【示例 2-9】用 continue、break 来做一个随机猜数字的游戏。先给定一个数值范围，系统在给定的范围内随机选取一个数，然后来猜这个随机数是多少，猜对了直接退出，猜错了系统则提示猜的数字与随机数相比是大了还是小了。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi guessNum.py
```

guessNum.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import random

class GuessNum(object):
    '''这个类用于猜随机数 '''
    def __init__(self):
        print(u"随机产生一个 0-100 的随机数")
        self.num = random.randint(0,101)
        self.guess()

    def guess(self):
        i = 0
        while True:
```

```
            print(u"猜这个随机数，0-100")
            strNum = raw_input("输入你猜的数字:")
            i += 1
            try:
                print("****************")
                if int(strNum) < self.num:
                    print(u"你猜得太小了")
                    continue
                elif int(strNum) > self.num:
                    print(u"你猜得太大了")
                    continue
                else:
                    print(u"你总算是猜对了")
                    print(u"你总共猜了%d 次" %i)
                    break
            except ValueError:
                print(u"只能输入数字，继续猜吧")
                continue
            print(u"如果没有 continue 或 break，就会显示这个，要不要试试？")


if __name__ == '__main__':
    gn = GuessNum()
```

输入:wq，保存 guessNum.py。guestNum 先指定了一个 1~100 的随机数。然后开始猜这个随机数是多少，一般来说猜 5 次左右就可以猜出来。如果能一次猜到这个随机数，有这么逆天的运气还是赶紧买几注彩票试试吧。执行命令：

```
python guestNum.py
```

得到的结果如图 2-15 所示。

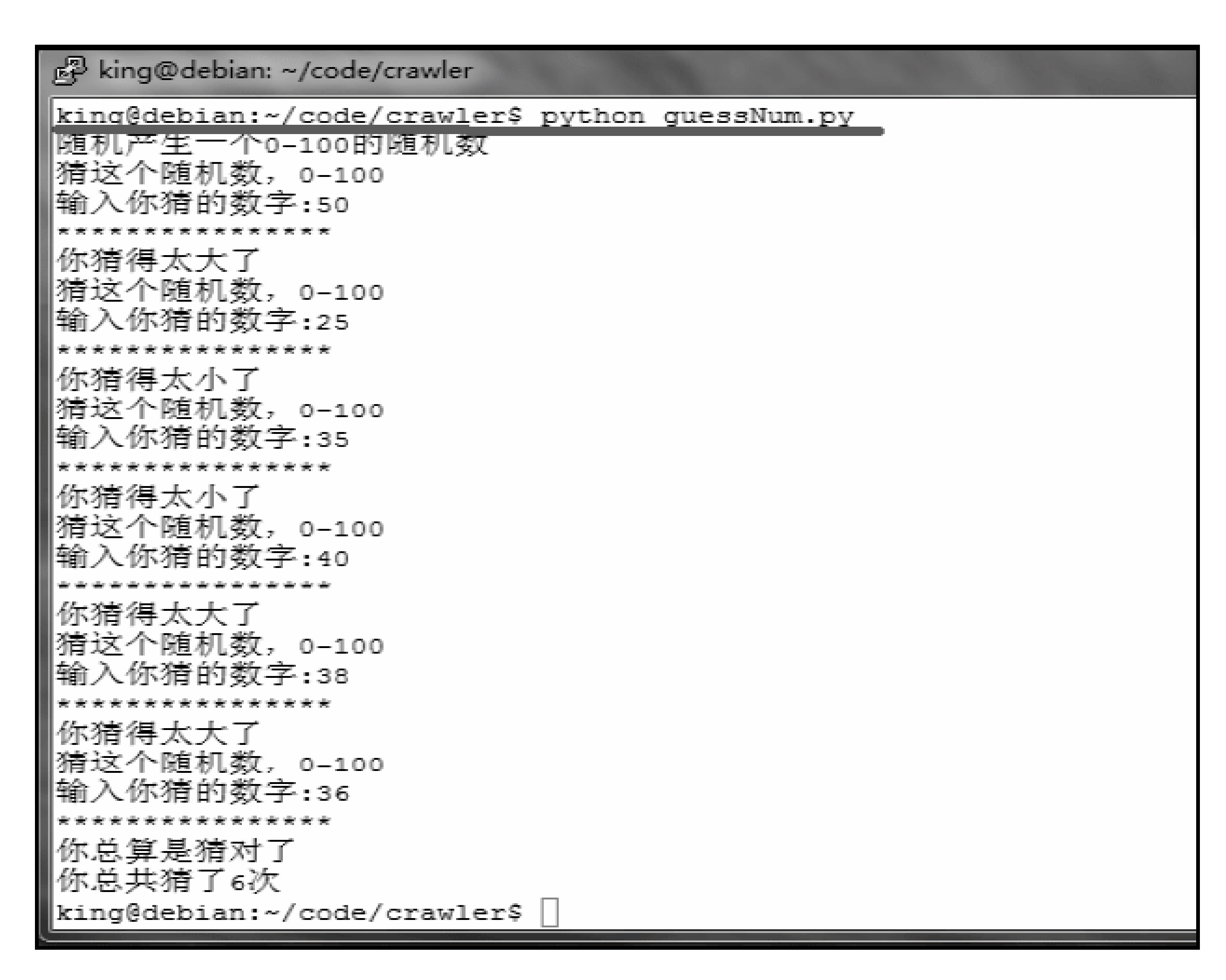


图 2-15　run guessNum.py

试一下，要猜多少次才会猜对这个随机数。



一般来说，纯粹只有循环而没有中断循环的情况很少见（特别是在 while 循环中）。大多都是配对出现的，所以熟悉了循环还必须掌握中断循环的方法。

2.2.5　异常处理——try except

要求输入的数据不符合要求，访问列表、元组下标超出范围，根据 key 访问字典中的 key 值却发现这个 key 不存在……编程时总会遇上种种意外。有些编程语言在碰到程序执行意外错误时，系统提示错误，然后退出程序。当然，Python 也是这样处理的，但不同的是 Python 还给出了其他的选择。

在 Python 中，用 try 来测试可能出现异常的语句，然后用 except 来处理可能出现的异常。try except 的表达形式如下：

```
try:
 语句
except [exception,[data…]]:
 Do something
except [exception,[data…]]:
 Do something
except [exception,[data…]]:
 Do something
```

意思是，尝试执行语句，如果出现某个异常则怎么做。因为同一个语句可能出现不同的异常，所以也会给出不同的解决方法。另外，try 还可以配以 else、finally 语句一起使用，不过这种情况比较少，有兴趣的朋友可以自行 google 用法。

【示例 2-10】以常见的输入数据异常为例，编写 testTryInput.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testTryInput.py
```

testTryInput.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

class TryInput(object):
    def __init__(self):
        self.len = 10
        self.numList = self.createList()
        self.getNum()

```

```
    def createList(self):
        print(u"创建一个长度为%d 的数字列表" %self.len)
        numL = []
        while len(numL) < 10:
            n = raw_input("请输入一个整数：")
            try:
                num = int(n)
            except ValueError:
                print(u"输入错误，要求是输入一个整数")
                continue
            numL.append(num)
            print(u"现在的列表为："),
            print(numL)
        return numL

    def getNum(self):
        print(u"当前列表为"),
        print(self.numList)
        inStr = None
        while inStr != 'EXIT':
            print(u"输入 EXIT 退出程序")
            inStr = raw_input("输入列表下标[-10,9]：")
            try:
                index = int(inStr)
                num = self.numList[index]
                print(u"列表中下标为%d 的值为%d" %(index,num))
            except ValueError:
                print(u"输入错误，列表下标是一个整数")
                continue
            except IndexError:
                print(u"下标太大，访问列表超出范围")
                continue


if __name__ == '__main__':
    ti = TryInput()
```

输入:wq，保存 testTryInput.py。testTryInput.py 目的是创建一个数字列表，在创建过程中尝试各种异常。执行命令：

```
python testTryInput.py
```

得到的结果如图 2-16 所示。

```
king@debian: ~/code/crawler
king@debian:~/code/crawler$ python testTryInput.py
创建一个长度为10的数字列表
请输入一个整数: x
输入错误, 要求是输入一个整数
请输入一个整数: 1
现在的列表为:  [1]
请输入一个整数: 2
现在的列表为:  [1, 2]
请输入一个整数: 3
现在的列表为:  [1, 2, 3]
请输入一个整数: 4
现在的列表为:  [1, 2, 3, 4]
请输入一个整数: 5
现在的列表为:  [1, 2, 3, 4, 5]
请输入一个整数: 6
现在的列表为:  [1, 2, 3, 4, 5, 6]
请输入一个整数: 7
现在的列表为:  [1, 2, 3, 4, 5, 6, 7]
请输入一个整数: 8
现在的列表为:  [1, 2, 3, 4, 5, 6, 7, 8]
请输入一个整数: 9
现在的列表为:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
请输入一个整数: 0
现在的列表为:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
当前列表为 [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
输入EXIT退出程序
输入列表下标[-10,9]: a
输入错误, 列表下标是一个整数
输入EXIT退出程序
输入列表下标[-10,9]: 100
下标太大, 访问列表超出范围
输入EXIT退出程序
输入列表下标[-10,9]: 8
列表中下标为8的值为9
输入EXIT退出程序
输入列表下标[-10,9]: EXIT
输入错误, 列表下标是一个整数
king@debian:~/code/crawler$
```

图 2-16　run testTryInput.py

这个程序就是针对输入出现的异常和访问列表越界的异常给出了解决方案。编程过程中总会遇上各种各样的异常。考虑周到一点，思维缜密一点，善用 try 一点，程序的健壮性就不止强一点点。

2.2.6　导入模块——import

个人看来，Python 最大的优点不是简单易学，而是其强大的模块功能。前面写的一个程序，后面就可以将它当成一个模块导入现在的程序，取其精华弃其糟粕地随意使用。最理想的情况是任何一个功能，只要写一次，以后所有人都可以任其调用。代码重用性高得可怕，而且 Python 还可以根据需求将 C、C++、Java 等程序作为模块，随意取用。这是为什么 Python 被称之为胶水语言的原因。

Python2 的标准模块（一般也叫 Python 标准库）是安装 Python 时自带的模块，具体请参考网页 https://docs.python.org/2.7/py-modindex.html。它包含了几乎所有的常用功能。如果觉得不够，没关系，可以用 pip 来安装第三方的模块，这个模块库就已经非常强大了。如果还不够，也没关系，还有强大的 github，全世界的 Pyther 在背后支持你。找到适用的功能程序导入到自己程序里就可以了。对别人程序极度不放心，非要自力更生也行，那就辛苦一下，自己写个程序做自己独有的模块吧。

模块导入的几种方式如下，可根据需要自行选择：

```
#同时导入多个模块
import module1[, module2[,... moduleN]
#导入模块中的某个函数、类、变量
from modname import name1[, name2[, ... nameN]]
#导入某个模块中所有
from modname import *
```

每次使用 print 打印时，总是同一个颜色。能不能使用不同的颜色打印呢？当然可以，第三方模块库里就有相关的模块。只需要使用 pip 安装即可，github 仔细找找应该也能找得到。在这里自力更生，自己动手写一个最符合自己要求的彩色打印的 print。

【示例 2-11】编写 testImportColorPrint.py，将它作为模块导入到其他的 python 程序中使用。打开 Putty，连接到 Linux，执行命令：

```
cd code/crawler
vi testImportColorPrint.py
```

testImportColorPrint.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import sys

class ColorPrint(object):  #带颜色的打印
    def __init__(self,color,msg):
        self.color = color
        self.msg = msg
        self.cPrint(self.color,self.msg)

    def cPrint(self,color,msg):
        colors = {
'black' :   '\033[1;30;47m',  #黑色
'red'   :   '\033[1;31;47m',  #红色
'green' :   '\033[1;32;47m',  #绿色
'yellow':   '\033[1;33;47m',  #黄色
'blue'  :   '\033[1;34;47m',  #蓝色
'white' :   '\033[1;37;47m'}  #白色
        if color not in colors.keys():
            print(u"输入的颜色暂时没有，按系统默认配置的颜色打印")
        else:
            print(u"输入的颜色有效,开始彩色打印")
            print(u"%s" %colors[color])
            print(msg)
```

```
            print(u"\033[0m")


if __name__ == '__main__':
    cp = ColorPrint(sys.argv[1],sys.argv[2])
```

输入:wq，保存 testImportColorPrint.py。这里只写入了黑色、红色、绿色、黄色、蓝色和白色这几种颜色。如需添加其他的颜色请自行 google 一下。执行命令：

```
python testImportColorPrint.py  black "I'm black"
python testImportColorPrint.py  red "I'm red"
python testImportColorPrint.py  green "I'm green"
python testImportColorPrint.py  yellow "I'm yellow"
python testImportColorPrint.py  blue "I'm blue"
python testImportColorPrint.py  white "I'm white"
```

得到的结果如图 2-17 所示。



图 2-17　run testImportColorPrint.py

彩色打印已经实现了（白色打印时因为背景色也是白色，所以显示不明显），下面是将 testImportColorPrint.py 当作模块导入到其他 Python 程序中供其使用。

【示例 2-12】无须太复杂，写个最简单的 testImport.py，只要能将 testImportColorPrint.py 当成模块导入使用即可。执行命令：

```
cd code/crawler
vi testImport.py
```

testImport.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

from testImportColorPrint import ColorPrint
#这里的testImportColorPrint模块就是从当前目录下导入的testImportColorPrint.py程序

if __name__ == '__main__':
    p_black = ColorPrint('black','I am black print')
    p_black = ColorPrint('red','I am red print')
    p_black = ColorPrint('green','I am green print')
    p_black = ColorPrint('yellow','I am yellow print')
    p_black = ColorPrint('blue','I am blue print')
    p_black = ColorPrint('white','I am white print')
```

输入:wq，保存 testImport.py。testImport.py 尝试调用 testImportColorPrint.py 脚本作为模块，调用该脚本的类放到自己的脚本中执行。执行命令：

```
python testImport.py
```

得到的结果如图 2-18 所示。

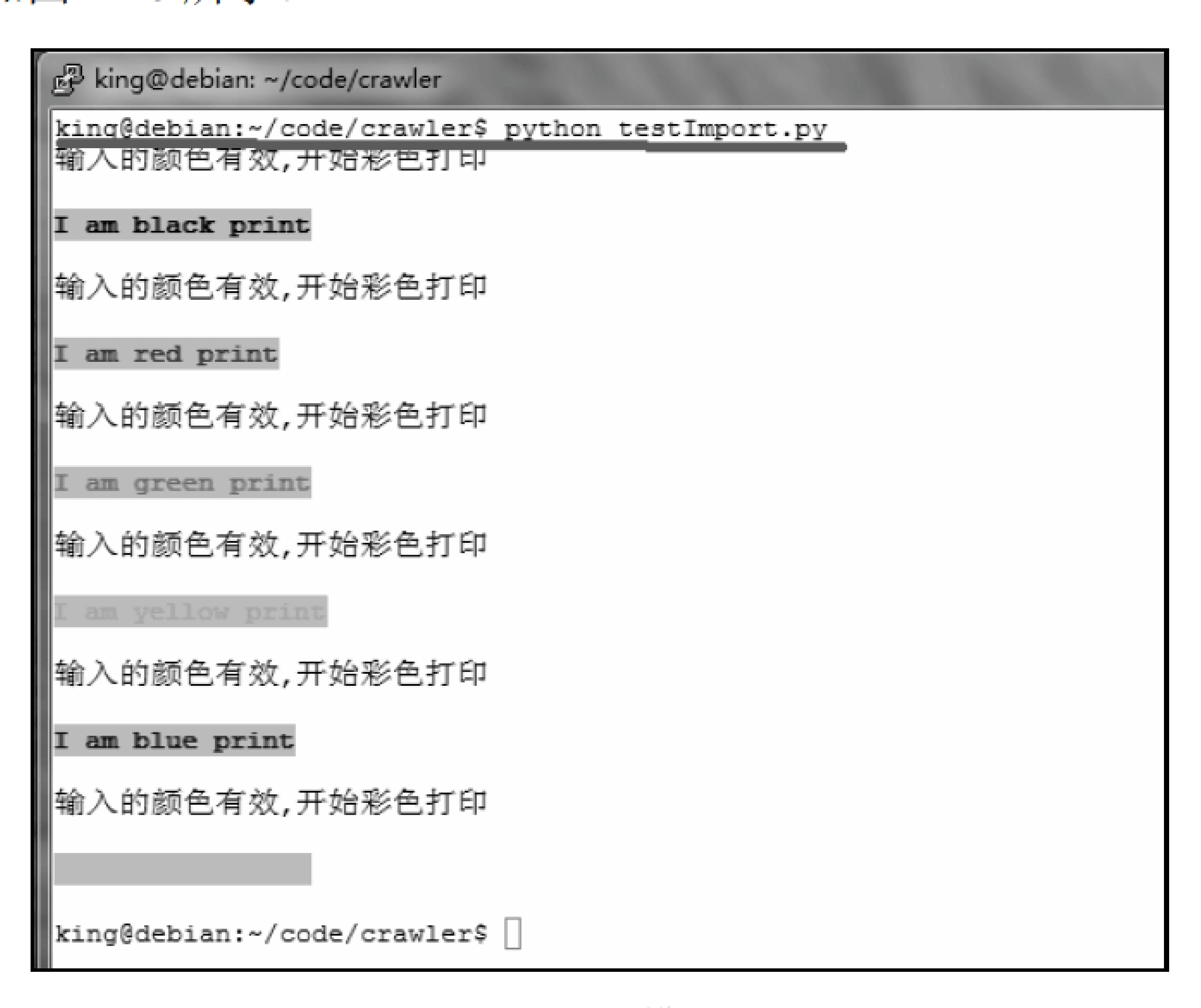


图 2-18　导入模块测试



将 Python 程序当成模块导入的先决条件是，这两个程序在同一目录下。或者将模块化的程序（这里就是 testImportColorPrint.py）路径加入到 Python 的系统路径中。是不是很简单呢？其实 Python 就是这么简单。

2.3 函数和类

C、C++、Java、Ruby、Perl、Lisp……在笔者所知的编程语言之中，所有的程序都是由函数和类组成的。可以说任何程序里面包含的不是函数就是类，Pyther 当然也不例外。

2.3.1　函数

曾经有一句非常出名的话是 In Unix Everything Is A File，在 Unix 中所有的一切都是文件。在这里可以借鉴一下，In Python Everything Is A Function，在 Python 程序中，所有的一切都是函数。这是典型的 C 语言写法，把所需的功能都写成一个一个的函数，然后由函数调用函数。依次类推，最终完成整个程序的功能。

还记得上节提过的暴力破解吗？不管用什么工具，暴力破解都少不了一个合适的字典（此字典非彼字典，这里的字典指的是一个包含密码的文件，也就是一个密码集，而不是 Python 的变量类型）。当然网上有很多的密码字典可供下载，但它们要么太大遍历一次需要太多的时间，要么没有针对性根本就不包含所需的密码。如果已知了一些可能是密码的字符串，完全可以根据已知条件用程序编写有针对性的字典出来，这样会节省很多时间。

【示例 2-13】现在来编写一个简单的程序 makePasswordFileFunction.py，创建一个有针对性的专用密码字典。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi makePasswordFileFunction.py
```

makePasswordFileFunction.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import os
import platform
import itertools
import time

def main():
    '''主程序 '''
    global rawList #原始数据列表
    rawList = []
    global denyList #非法单词列表
    denyList = [' ','','@']
    global pwList #最终的密码列表
    pwList = []
```

```
    global minLen #密码的最小长度
    minLen = 6
    global maxLen #密码的最大长度
    maxLen = 16
    global timeout
    timeout = 3
    global flag
    flag = 0
    run = {
'0':exit,  #退出
'1':getRawList,  #创建原始列表
'2':addDenyList,  #添加不可能出现的元素
'3':clearRawList,  #清空列表
'4':setRawList,  #原始列表排序
'5':modifyPasswordLen,  #修改最终的密码长度
'6':createPasswordList,  #创建最终的字典列表
'7':showPassword,  #显示密码
'8':createPasswordFile  #创建密码文件
}

    while True:
        mainMenu()
        op = raw_input('输入选项:')
        if op in map(str,range(len(run))):
            run.get(op)()
        else:
            tipMainMenuInputError()
            continue

def mainMenu():
    '''主菜单 '''
    global denyList
    global rawList
    global pwList
    global flag
    clear()
    print(u'||'),
    print(u'='*40),
    print(u'||')
    print(u'|| 0:退出程序')
    print(u'|| 1:输入密码原始字符串')
    print(u'|| 2:添加非法字符到列表')
    print(u'|| 3:清空原始密码列表')
```

```
    print(u'|| 4:整理原始密码列表')
    print(u'|| 5:改变默认密码长度(%d-%d)' %(minLen,maxLen))
    print(u'|| 6:创建密码列表')
    print(u'|| 7:显示所有密码')
    print(u'|| 8:创建字典文件')
    print(u'||'),
    print(u'='*40),
    print(u'||')
    print(u'当前非法字符为: %s' %denyList)
    print(u'当前原始密码元素为: %s' %rawList)
    print(u'共有密码%d 个' %len(pwList))
    if flag:
        print(u"已在当前目录创建密码文件 dic.txt")
    else:
        print(u"尚未创建密码文件")

def clear():
    '''清屏函数 '''
    OS = platform.system()
    if (OS == u'Windows'):
        os.system('cls')
    else:
        os.system('clear')

def tipMainMenuInputError():
    '''错误提示 '''
    clear()
    print(u"只能输入 0-7 的整数,等待%d 秒后重新输入" %timeout)
    time.sleep(timeout)

def getRawList():
    '''获取原始数据列表 '''
    clear()
    global denyList
    global rawList
    print(u"输入回车后直接退出")
    print(u"当前原始密码列表为:%s" %rawList)
    st = None
    while not st == '':
        st = raw_input("请输入密码元素字符串:")
        if st in denyList:
            print(u"这个字符串是预先设定的非法字符串")
            continue
```

```
        else:
            rawList.append(st)
            clear()
            print(u"输入回车后直接退出")
            print(u"当前原始密码列表为:%s" %rawList)

def addDenyList():
    '''添加非法词 '''
    clear()
    global denyList
    print(u"输入回车后直接退出")
    print(u"当前非法字符为:%s" %denyList)
    st = None
    while not st == '':
        st = raw_input("请输入需要添加的非法字符串:")
        denyList.append(st)
        clear()
        print(u"输入回车后直接退出")
        print(u"当前非法字符列表为:%s" %denyList)

def clearRawList():
    '''清空原始数据列表 '''
    global rawList
    rawList = []

def setRawList():
    '''整理原始数据列表 '''
    global rawList
    global denyList
    a = set(rawList)
    b = set(denyList)
    rawList = []
    for str in set(a - b):
        rawList.append(str)

def modifyPasswordLen():
    '''修改默认密码的长度 '''
    clear()
    global maxLen
    global minLen
    while True:
        print(u"当前密码长度为%d-%d" %(minLen,maxLen))
        min = raw_input("请输入密码最小长度:")
```

```
        max = raw_input("请输入密码最大长度:")
        try:
            minLen = int(min)
            maxLen = int(max)
        except ValueError:
            print(u"密码长度只能输入数字[6-18]")
            break
        if minLen not in xrange(6,19) or  maxLen not in xrange(6,19):
            print(u"密码长度只能输入数字[6-18]")
            minLen = 6
            maxLen = 16
            continue
        if minLen == maxLen:
            res = raw_input("确定将密码长度设定为%d 吗?(Yy/Nn)" %minLen)
            if res not in list('yYnN'):
                print(u"输入错误，请重新输入")
                continue
            elif res in list('yY'):
                print(u"好吧，你确定就好")
                break
            else:
                print(u"给个机会，改一下吧")
                continue
        elif minLen > maxLen:
            print(u"最小长度比最大长度还大，可能吗？请重新输入")
            minLen = 6
            maxLen = 16
            continue
        else:
            print(u"设置完毕，等待%d 秒后回主菜单" %timeout)
            time.sleep(timeout)
            break

def createPasswordList():
    '''创建密码列表 '''
    global rawList
    global pwList
    global maxLen
    global minLen
    titleList = []
    swapcaseList = []
    for st in rawList:
        swapcaseList.append(st.swapcase())
```

```
            titleList.append(st.title())
    sub1 = []
    sub2 = []
    for st in set(rawList + titleList + swapcaseList):
        sub1.append(st)
    for i in xrange(2,len(sub1) + 1):
        sub2 += list(itertools.permutations(sub1,i))
    for tup in sub2:
        PW = ''
        for subPW in tup:
            PW += subPW
        if len(PW) in xrange(minLen,maxLen + 1):
            pwList.append(PW)
        else:
            pass

def showPassword():
    '''显示创建的密码 '''
    global pwList
    global timeout
    for i in xrange(len(pwList)):
        if i%4 == 0:
            print("%s\n" %pwList[i])
        else:
            print("%s\t" %pwList[i]),
    print('\n')
    print(u"显示%d 秒，回到主菜单" %timeout)
    time.sleep(timeout)

def createPasswordFile():
    '''创建密码字典文件 '''
    global flag
    global pwList
    print(u"当前目录下创建字典文件:dic.txt")
    time.sleep(timeout)
    with open('./dic.txt','w+') as fp:
        for PW in pwList:
            fp.write(PW)
            fp.write('\n')
    flag = 1


if __name__ == '__main__':
```

```
main()
```

输入:wq，保存 makePasswordFileFunction.py。makePasswordFileFunction.py 稍微复杂一点点，它的作用就是根据用户输入的“密码元素”来创建一个字典列表。该脚本将输入的元素根据一定的规则修改、添加后当作新元素添加到元素列表中去。最后将元素列表排列组合得到最后的字典列表。执行命令：

```
python makePasswordFileFunction.py
```

得到的结果如图 2-19 所示。

```
king@debian: ~/code/crawler
|| ======================================== ||
|| 0:退出程序
|| 1:输入密码原始字符串
|| 2:添加非法字符到列表
|| 3:清空原始密码列表
|| 4:整理原始密码列表
|| 5:改变默认密码长度(6-16)
|| 6:创建密码列表
|| 7:显示所有密码
|| 8:创建字典文件
|| ======================================== ||
当前非法字符为: [' ', '', '@']
当前原始密码元素为: []
共有密码0个
尚未创建密码文件
输入选项:
```

图 2-19　run makePasswordFileFunction.py

纯 C 语言的写法好处就是关系简单明了，函数调用一目了然。但如果调用的函数过多，就难免有些混乱了。简单功能的程序还无妨，稍大一点项目就有些吃力了。



不要添加太多的“密码元素”，这个程序只是利用了 Python 的模块，没有优化算法。如果输入的“密码元素”超过了 20 个，那么创建密码字典的时间会非常的长。

2.3.2　类

既然有了 In Python Everything Is A Fcuntion，当然会有 In Python Everything Is A Class。这种 C++的写法就是把所有相似的功能都封装到一个类里。最理想的情况是一个程序只有一个主程序，然后在主程序里实例化类。

【示例 2-14】还是以编写密码字典为例，将 makePasswordFileFunction.py 改编成 makePasswordFileClass.py。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi makePasswordFileClass.py
```

makePasswordFileClass.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'
```

```

import os
import platform
import itertools
import time

class MakePassword(object):
    def __init__(self):
        self.rawList = []
        self.denyList = ['',' ','@']
        self.pwList = []
        self.minLen = 6
        self.maxLen = 16
        self.timeout = 3
        self.flag = 0
        self.run = {
'0':exit,
'1':self.getRawList,
'2':self.addDenyList,
'3':self.clearRawList,
'4':self.setRawList,
'5':self.modifyPasswordLen,
'6':self.createPasswordList,
'7':self.showPassword,
'8':self.createPasswordFile
}
        self.main()

    def main(self):
        while True:
            self.mainMenu()
            op = raw_input('输入选项:')
            if op in map(str,range(len(self.run))):
                self.run.get(op)()
            else:
                self.tipMainMenuInputError()
                continue

    def mainMenu(self):
        self.clear()
        print(u'||'),
        print(u'='*40),
        print(u'||')
```

```
        print(u'|| 0:退出程序')
        print(u'|| 1:输入密码原始字符串')
        print(u'|| 2:添加非法字符到列表')
        print(u'|| 3:清空原始密码列表')
        print(u'|| 4:整理原始密码列表')
        print(u'|| 5:改变默认密码长度(%d-%d)' %(self.minLen,self.maxLen))
        print(u'|| 6:创建密码列表')
        print(u'|| 7:显示所有密码')
        print(u'|| 8:创建字典文件')
        print(u'||'),
        print(u'='*40),
        print(u'||')
        print(u'当前非法字符为：%s' %self.denyList)
        print(u'当前原始密码元素为：%s' %self.rawList)
        print(u'共有密码%d 个' %len(self.pwList))
        if self.flag:
            print(u"已在当前目录创建密码文件 dic.txt")
        else:
            print(u"尚未创建密码文件")

    def clear(self):
        OS = platform.system()
        if (OS == u'Windows'):
            os.system('cls')
        else:
            os.system('clear')

    def tipMainMenuInputError(self):
        self.clear()
        print(u"只能输入 0-7 的整数,等待%d 秒后重新输入" %timeout)
        time.sleep(timeout)

    def getRawList(self):
        self.clear()
        print(u"输入回车后直接退出")
        print(u"当前原始密码列表为:%s" %self.rawList)
        st = None
        while not st == '':
            st = raw_input("请输入密码元素字符串:")
            if st in self.denyList:
                print(u"这个字符串是预先设定的非法字符串")
                continue
            else:
```

```
            self.rawList.append(st)
            self.clear()
            print(u"输入回车后直接退出")
            print(u"当前原始密码列表为:%s" %self.rawList)

    def addDenyList(self):
        self.clear()
        print(u"输入回车后直接退出")
        print(u"当前非法字符为:%s" %self.denyList)
        st = None
        while not st == '':
            st = raw_input("请输入需要添加的非法字符串:")
            self.denyList.append(st)
            self.clear()
            print(u"输入回车后直接退出")
            print(u"当前非法字符列表为:%s" %self.denyList)

    def clearRawList(self):
        self.rawList = []

    def setRawList(self):
        a = set(self.rawList)
        b = set(self.denyList)
        self.rawList = []
        for str in set(a - b):
            self.rawList.append(str)

    def modifyPasswordLen(self):
        self.clear()
        while True:
            print(u"当前密码长度为%d-%d" %(self.minLen,self.maxLen))
            min = raw_input("请输入密码最小长度:")
            max = raw_input("请输入密码最大长度:")
            try:
                self.minLen = int(min)
                self.maxLen = int(max)
            except ValueError:
                print(u"密码长度只能输入数字[6-18]")
                break
            if self.minLen not in xrange(6,19) or  self.maxLen not in xrange(6,19):
                print(u"密码长度只能输入数字[6-18]")
                self.minLen = 6
```

```
                self.maxLen = 16
                continue
            if self.minLen == self.maxLen:
                res = raw_input("确定将密码长度设定为%d 吗?(Yy/Nn)" %self.minLen)
                if res not in list('yYnN'):
                    print(u"输入错误，请重新输入")
                    continue
                elif res in list('yY'):
                    print(u"好吧，你确定就好")
                    break
                else:
                    print(u"给个机会，改一下吧")
                    continue
            elif self.minLen > self.maxLen:
                print(u"最小长度比最大长度还大，可能吗？请重新输入")
                self.minLen = 6
                self.maxLen = 16
                continue
            else:
                print(u"设置完毕，等待%d 秒后回主菜单" %self.timeout)
                time.sleep(self.timeout)
                break

    def createPasswordList(self):
        titleList = []
        swapcaseList = []
        for st in self.rawList:
            swapcaseList.append(st.swapcase())
            titleList.append(st.title())
        sub1 = []
        sub2 = []
        for st in set(self.rawList + titleList + swapcaseList):
            sub1.append(st)
        for i in xrange(2,len(sub1) + 1):
            sub2 += list(itertools.permutations(sub1,i))
        for tup in sub2:
            PW = ''
            for subPW in tup:
                PW += subPW
            if len(PW) in xrange(self.minLen,self.maxLen + 1):
                self.pwList.append(PW)
            else:
```

```
                pass

    def showPassword(self):
        for i in xrange(len(self.pwList)):
            if i%4 == 0:
                print("%s\n" %self.pwList[i])
            else:
                print("%s\t" %self.pwList[i]),
        print('\n')
        print(u"显示%d 秒，回到主菜单" %self.timeout)
        time.sleep(self.timeout)

    def createPasswordFile(self):
        print(u"当前目录下创建字典文件:dic.txt")
        time.sleep(self.timeout)
        with open('./dic.txt','w+') as fp:
            for PW in self.pwList:
                fp.write(PW)
                fp.write('\n')
        self.flag = 1


if __name__ == '__main__':
    mp = MakePassword()
```

输入 :wq，保存 makePasswordFileClass.py。makePasswordFileClass.py 和 makePasswordFileFunction.py 实质上没什么区别，只是一个使用的是 C 语言风格的函数调用，一个使用的是 C++风格的类实例化。执行命令：

```
python makePasswordFileClass.py
```

得到的结果如图 2-20 所示。

```
king@debian: ~/code/crawler
|| ======================================= ||
|| 0:退出程序
|| 1:输入密码原始字符串
|| 2:添加非法字符到列表
|| 3:清空原始密码列表
|| 4:整理原始密码列表
|| 5:改变默认密码长度(6-16)
|| 6:创建密码列表
|| 7:显示所有密码
|| 8:创建字典文件
|| ======================================= ||
当前非法字符为: [' ', '', '@']
当前原始密码元素为: []
共有密码0个
尚未创建密码文件
输入选项:
```

图 2-20　run makePasswordFileClass.py

执行结果完全一样。这种 C++的写法好处就是调用过程简单，不再关心类具体的实现过程，只需要调用其功能即可；但随之而来就是类的继承、函数重载等麻烦。这种写法在写大项目时可能非常有用。个人写小程序也行，那就没那么多优势了。



这个密码还有个问题就是在创建密码文件前并没有估算磁盘剩余空间是否足够。一般的解决办法是先估算密码文件的大小，然后创建一个大小相同的空文件。能创建成功则继续运行程序，不能则抛出异常。

2.4 Python 代码格式

Python 是一门新兴的编程语言，在格式方面与其他大众语言相差不大，但也有它独特之处，尤其是代码缩进。在其他的编程语言中，代码缩进大多是为了美观，程序、函数的开始结束都是由花括号来控制的。而在 Python 中却不一样，程序、代码块的开始结束都是由缩进来控制的。所以，首先要熟悉的就是 Python 的代码缩进。

2.4.1 Python 代码缩进

Python 的缩进一般来说是 4 个空格，先严格按照这种缩进方法来写个测试代码。

```
Class TestBlank(object):
----|----|def __init__(self):
----|----|----|----|self.timeout = 3
----|----|----|----|self.url = ''
..............................
```

以上的代码中----|代表 4 个空格。这才写了个开头就得 40 个空格。要是 Python 只能这样写，那笔者还是宁愿选择 C 或者 C++。好在还有备用方案，可以用 Tab 键来替代 4 个空格。这样的好处就是少按了很多次空格，坏处就是代码不好移植。在这台电脑上可以运行的程序，换台电脑可能就无法直接使用了。

既然变通了，那就变通到底好了。实际上这也是目前流行的做法，在自己的代码编辑器上将 Tab 键设置成 4 个空格就可以了。比如 Windows 下的 notepad++就可以在“设置 | 首选项 | 语言”菜单中选中以空格替代。其他的 Python IDE 中都有类似的设定，自行摸索一下就可以了。Linux 下一般用的都是 vi，那就更加简单了。在/etc/vim/vimrc 或者~/.vim/vimrc 中添加代码：

```
set ts=4
set expandtab
```



有的 vi 默认将 tabstop 定义成了 8 个空格。

Python 每行代码前的缩进都有语法和逻辑上的意义。在严格要求的代码缩进之下，代码非常整齐规范，赏心悦目，提高了可读性，在一定程度上也提高了可维护性。

至于 Python 的缩进规则很简单。简单说就是，同一代码块纵向对齐。同级别函数（不存在调用关系的）纵向对齐，每次对齐都是 4 个空格的倍数。如果违反这些规则，Python 是不会工作的，只会给一条冷冰冰的异常通知：SyntaxError: invalid syntax。

2.4.2 Python 命名规则

对于给类、函数、变量取名，只要不违法命名规则，取任何名字都是可以的。要是不明白类、函数、变量的作用不是还有注释吗？的确是这样的。但如果能“望名生义”那又何必去添加多余的注释呢？另外，统一的命名法也令程序看起来赏心悦目。编写代码不能以书法让人愉悦，那就以名字和格式让人愉悦吧。

1．匈牙利命名法

据说匈牙利命名法是一位叫 Charles Simonyi 的匈牙利程序员发明的，后来他在微软待了几年，于是这种命名法就通过微软的各种产品和文档资料向世界传播开了。这种命名法的出发点是把变量名按：属性+类型+对象描述的顺序组合起来，以使程序员定义变量时对变量的类型和其他属性有直观的了解。

这种命名方法的确很好。可惜的是，Python 的参数并不像 C、C++、Java 一样，声明变量无须指定变量类型。而且在没用到 Python GUI 编程前也不会遇到属性、对象什么的，所以这种命名法还是等到使用 GUI 编程时再使用吧。

2．驼峰命名法

骆驼式命名法（Camel-Case）又称驼峰命名法，是计算机程序编写时的一套命名规则（惯例）。正如它的名称 CamelCase 所表示的那样，是指混合使用大小写字母来构成变量和函数的名字。

骆驼式命名法就是当变量名或函式名是由一个或多个单词连在一起，而构成的唯一识别字时，第一个单词以小写字母开始；第二个单词的首字母大写或每一个单词的首字母都采用大写字母，例如：myFirstName、myLastName，这样的变量名看上去就像骆驼峰一样此起彼伏，故得名。驼峰命名法又分为小驼峰命名法和大驼峰命名法。

变量和函数一般用小驼峰法标识。驼峰法的意思是：除第一个单词之外，其他单词首字母大写。譬如：

```
def getUrl
urlSrc = u'http://ww.baidu.com'
```

变量 urlSrc 第一个单词是全部小写，后面的单词首字母大写。

相比小驼峰法，大驼峰法把第一个单词的首字母也大写了，有时它也被称之为帕斯卡（pascal）命名法。常用于类名。譬如：

```
Class MyLog(object):
```

3．Guido 推荐的命名规则

Python 之父 Guido 推荐在 python 中使用的命名方法。如表 2-1 所示。

表 2-1 PythonName

Type	Public	Internal
Modules	low_with_under	_lower_with_under
Packages	low_with_under	
Classes	CapWords	_CapWords
Exceptions	CapWords	
Functions	lower_with_under()	_lower_with_under()
Global/Class Constants	CAPS_TITH_UNDER	_CAPS_WITH_UNDER
Global/Class Variables	low_with_under	_lower_with_under
Instance Variables	low_with_under	_lower_with_under (protected) or __lower_with_under (private)
Method Names	low_with_under()	_lower_with_under() (protected) or __lower_with_under() (private)
Function/Method Parameters	low_with_under	
Local Variables	low_with_under	

命名约定如下：

- 所谓“内部（Internal）”表示仅模块内可用，或者在类内是保护或私有的。
- 用单下划线(_)开头表示模块变量或函数是 protected 的（使用 import * from 时不会包含）。
- 用双下划线(__)开头的实例变量或方法表示类内私有。
- 将相关的类和顶级函数放在同一个模块里，不像 Java，没必要限制一个类一个模块。
- 对类名使用大写字母开头的单词（如 CapWords，即 Pascal 风格），但是模块名应该用小写加下划线的方式（如 lower_with_under.py），尽管已经有很多现存的模块使用类似于 CapWords.py 这样的命名，但现在已经不鼓励这样做，因为如果模块名碰巧和类名一致，这会让人困扰。

以上的三种命名规则，可以任选一种或者组合使用，并没有强制要求。理论上来说，选择 Python 推荐的命名规则比较好，这也是 Google 推荐的 Python 命名规则。但这事也可以这样，你拿着饮品质问店家为什么上面写着建议零售价 3 块，你非要卖我 5 块，店家可以理直气壮地回复，我不接受这个建议。你也可以不接受 Google 建议，选择自己喜欢的命名方法就

好，只要自己能看懂，交流无障碍就可以了。

2.4.3 Python 代码注释

一个好的程序员，为代码添加注释是编码时必须要做的，但要确保注释中要说明的都是重要的事情，让其他人看一眼就知道是干什么用的。注释在任何语言的代码中都非常重要，没有哪一种语言是完全不需要注释的。在 Python 中，注释还有其他的作用。Python 中的注释分为特殊注释、单行注释和多行注释。

1. Python 特殊注释

```
#!/usr/bin/python env
#-*- coding:utf-8 -*-
```

在所有的 Python 代码开头都有这两句（在 Windows 中写代码可以不用第一行注释，但为了移植方便，让程序能直接在 Linux 下运行还是加上这行比较好）。

以上特殊注释的第一行目的是指明 Python 编译器位置。第二行则指定了该程序使用的字符编码。指定字符编码还可以写成：

```
#coding=utf-8
```

只能写成这两种形式，字符编码可以任选，但格式一个字符都不可以错。

2. Python 单行注释

单行注释很简单。不管在代码的任何位置，只要是#之后的都是注释。但仅限于本行之内，不得换行。单行注释的代码如下：

```
self.timeout = 5  #网络超时时间
self.fileName = './todayMoive.txt'  #保存文件的位置
```

单行注释不需要刻意地对齐，避免出现 SyntaxError: invalid syntax 的异常。

3. Python 多行注释

Python 中的多行注释采取的是三个单引号’’’或者三个双引号”””。如果多行注释紧跟着定义类或者定义函数之后则自动变成了该类或者函数的 doc string。什么是 doc string 呢？简单地说就是模块、类、函数的功能注释。

【示例 2-15】写个简单的例子，一试就清楚了。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testAnnotation.py
```

testAnnotation.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
```

```
 3 __author__ = 'hstking hstking@hotmail.com'
 4
 5 class Annotation(object):
 6     '''这是一个用户示范注释的类，
 7 多行注释如果在类或者函数的定义之后，
 8 将被默认成 doc string。
 9 这里注释的是该类的功能性说明'''
10     def __init__(self):
11         self.run()
12
13     def run(self):
14         """函数里的 doc string，
15 这里注释的是该函数的功能性说明
16 注释用单引号和双引号没有任何区别 """
17         x = 333 #定义了一个 int 类型的变量 x
18         print('x = %d' %x)
19         '''好了，这里是单纯的注释了。可以注释多行，当然也可以注释单行了 '''
20
21
22 if __name__ == '__main__':
23     a = Annotation()
```

在 testAnnotation.py 中，第 17 行使用的是单行注释，第 19 行使用的是多行注释。其他的则是类和函数的 doc string。至于 doc string 怎么显示也挺简单的。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
python
import testAnnotation
print testAnnotation.Annotation.__doc__
print testAnnotation.Annotation.run.__doc__
```

执行结果如图 2-21 所示。

```
king@debian: ~/code/crawler
king@debian:~$ cd code/crawler
king@debian:~/code/crawler$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import testAnnotation
>>> print testAnnotation.Annotation.__doc__
这是一个用户示范注释的类，
多行注释如果在类或者函数的定义之后，
将被默认成doc string。
这里注释的是该类的功能性说明
>>> print testAnnotation.Annotation.run.__doc__
函数里的doc string，
这里注释的是该函数的功能性说明
注释用单引号和双引号没有任何区别
>>>
```

图 2-21　注释&doc string

注释就介绍到这里。在编程时不加入注释，当时可能没什么问题，待到以后维护代码时就会发现那是多么痛苦的事情。当然，如果打死都不做那个维护代码的人，那就无所谓什么注释不注释了。

2.5 Python 调试

调试是 Python 编程中非常重要的一环。程序出现什么问题，查看抛出的异常。或者处处加 print 和 log 找出错误点，再慢慢地反推，是可以找到问题解决问题的。但是有更简单的方法为什么非得舍易取难呢？

在 Linux 和 Windows 平台有很多的第三方调试工具，一般的 Python IDE 基本也自带了调试工具。工具太多了反而不好选择，而且也不是随手都能找到第三方调试工具的。这里仅示范手头上必定有的，Python 自带的调试工具。其他的第三方调试工具都大同小异，熟悉了最简单的，其他的也就无师自通了。

2.5.1 Windows 下 IDLE 调试

先写个简单的程序来做示例。既然是调试，最好的选择莫过于多次调用函数的阶乘了，这个程序简单又明显，适合用来做示例。打开 IDLE，单击菜单栏的 File|New File。创建一个新文档。编辑代码如图 2-22 所示。

testWinDebugFactorial.py - C:\Users\king\Desktop\testWinDebugFactorial.py (...

File Edit Format Run Options Window Help

```
#!/usr/bin/env python
#-*- coding:GBK -*-

def fac(n):
    if n==1 or n==0:
        return 1
    else:
        return n*fac(n-1)

def main():
    print('这是一个求阶乘的程序\n')
    n = raw_input('请输入一个正整数:')
    try:
        n = int(n)
    except ValueError:
        print('输入错误，要求输入一个正整数，退出重来吧。')
    print('%d! = %d' %(n,fac(n)))


if __name__ == '__main__':
    main()
```

图 2-22 testWinDebugFactorial.py

单击菜单栏 File|Save As，选择保存位置后将文件保存为 testWinDebugFactorial.py。下面开始调试 testWinDebugFactorial.py。

单击 IDLE 菜单栏的 Run|Python Shell，打开 Python Shell，如图 2-23 所示。

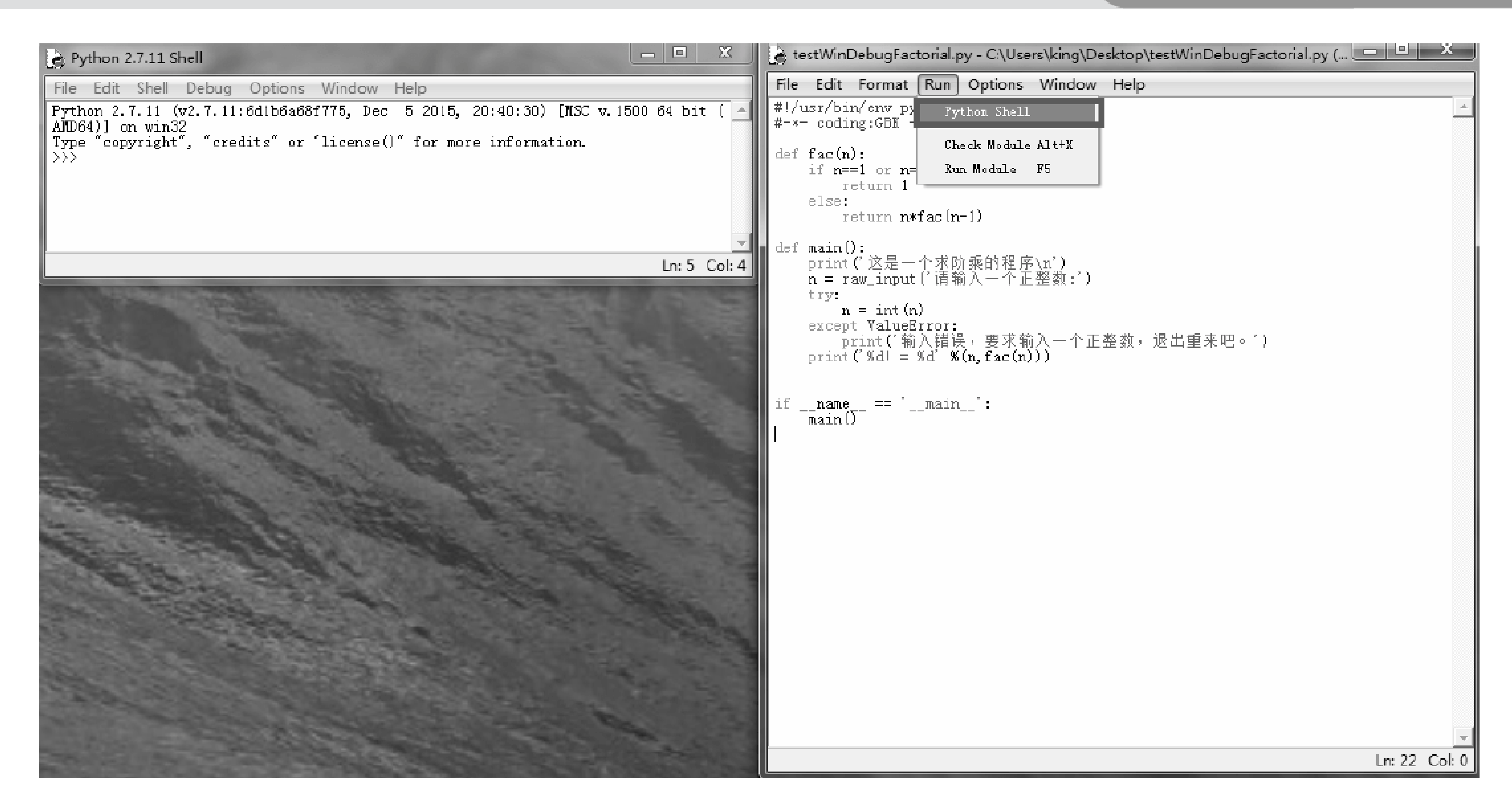


图 2-23　打开 Python Shell

单击 Python Shell 菜单栏的 Debug|Debugger。打开 Debug Control 窗口，如图 2-24 所示。

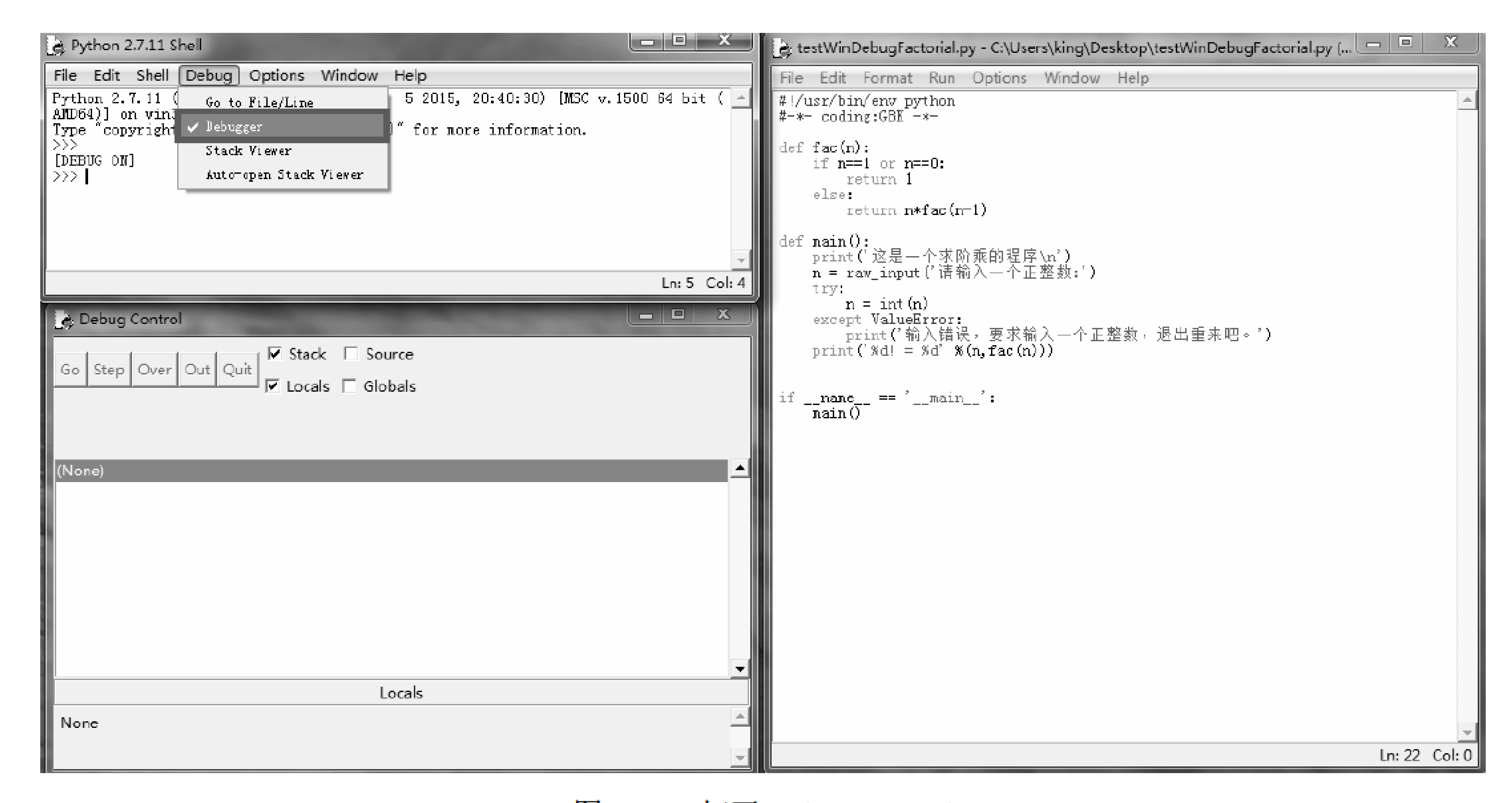


图 2-24　打开 Debug Control

然后在 IDLE 窗口为代码添加断点。所谓断点简单地说就是调试程序时需要停顿的位置。一般在函数的入口、参数变化的行添加。这里只在 fac 函数入口添加一个断点。单击 fac 函数入口行，再右击弹出菜单，选择 Set Breakpoing，如图 2-25 所示。

```
testWinDebugFactorial.py - C:\Users\king\Desktop\testWinDebugFactorial.py (...
File  Edit  Format  Run  Options  Window  Help
#!/usr/bin/env python
#-*- coding:GBK -*-

def fac(n):
    if r
    else
                Cut
                Copy
                Paste
def mai         Set Breakpoint
    pri         Clear Breakpoint   程序\n')
    n =                            个正整数:')
    try:
        n = int(n)
    except ValueError:
        print('输入错误，要求输入一个正整数，退出重来吧。')
    print('%d! = %d' %(n,fac(n)))


if __name__ == '__main__':
    main()
```

图 2-25　设置断点

现在可以开始运行调试程序了，单击 IDLE 窗口菜单栏的 Run|Run Module，如图 2-26 所示。

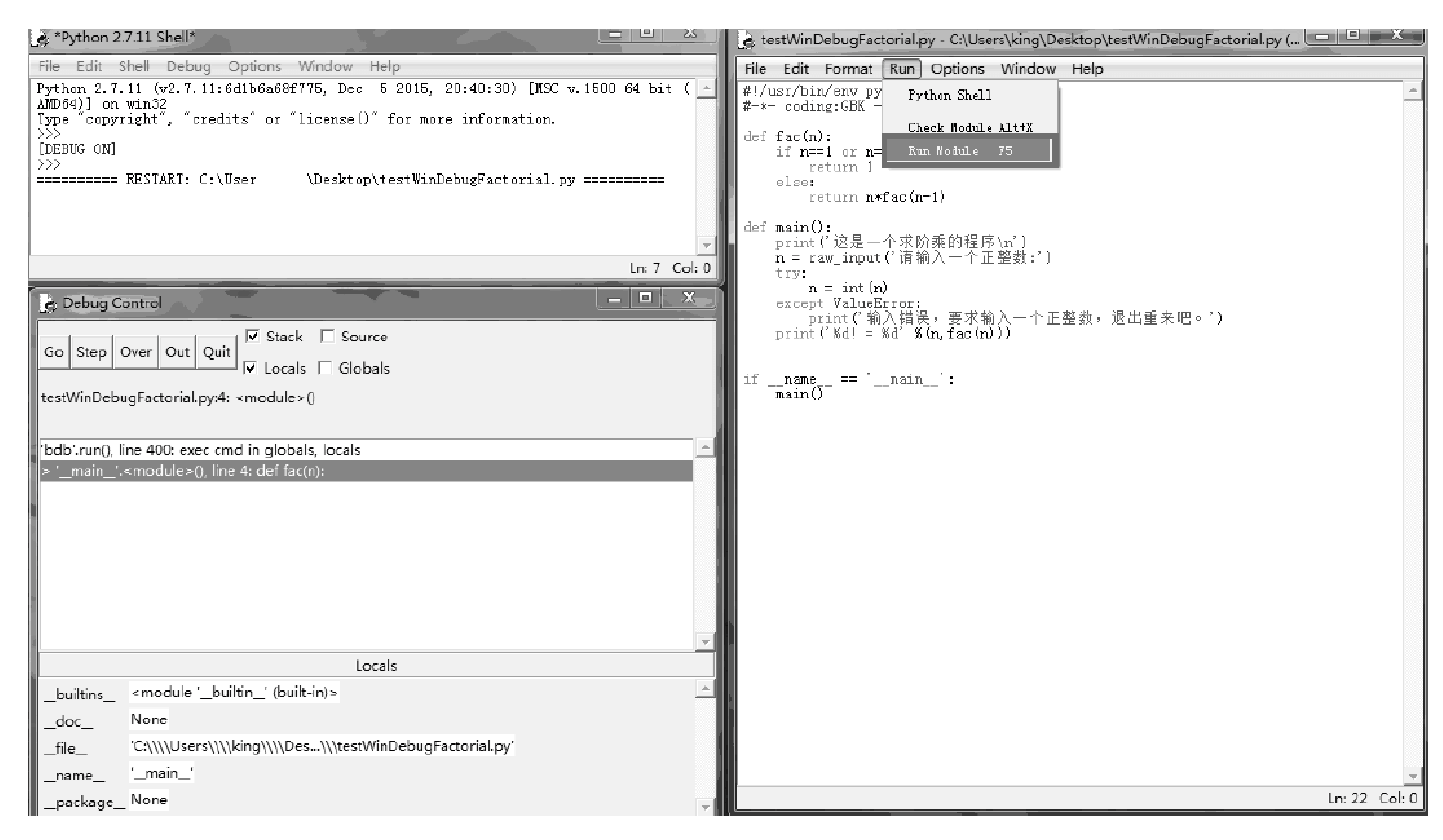


图 2-26　运行调试程序

单击 Debug Control 窗口的 Go 按钮，开始运行程序，然后单击 Debug Control 窗口的 Step 按钮，逐步运行程序。如果需跳出循环或者跳出函数，则单击 Debug Control 窗口的 Out 按钮。Debug Control 窗口中的 Stack 检查框显示的是程序当前运行位置，Locals 检查框显示的是当前变量的值，如图 2-27 所示。



图 2-27　Debug Control

通过 Debug 调试很容易发现程序中的错误之处。虽然这个 Debug 工具比较简陋，但基本功能都还齐全，算是比较好用的一款 Debug 工具了。

2.5.2　Linux 下 pdb 调试

Linux 下 Python 调试工具也很多，但最简单、最方便的可能就是 pdb 了。pdb 功能齐全，使用方便，使用过 gdb 的朋友会对它非常熟悉，它们的命令几乎是一模一样的。先写个示范程序，用 pdb 调试一下。

【示例 2-16】打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testLinuxBugListExtremum.py
```

testLinuxBugListExtremum.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import cls
import time

def getList():
    #构建一个纯数字列表
    numList = []
```

```
    num = 'q'
    while num:
        cls.clear()
        print numList
        print(u'结束构建列表，请按回车')
        num = raw_input('请输入一个整数：')
        if num == '':
            break
        try:
            num = int(num)
        except ValueError:
            print(u'要求输入整数，请重新输入')
            time.sleep(1)
            continue
        numList.append(num)
    return numList

def getMaxNum(List):
    #获取列表中最大值
    num = List[0]
    for i in List[1:]:
        if num <= i:
            num = i
    return num

def getMinNum(List):
    #获取列表中最小值
    num = List[0]
    for i in List[1:]:
        if num >= i:
            num = i
    return num


if __name__ == '__main__':
    numList = getList()
    maxNum = getMaxNum(numList)
    print(u'列表中最大值为:%d' %maxNum)
    minNum = getMinNum(numList)
    print(u'列表中最小值为:%d' %minNum)
```

testLinuxBugListExtremum.py 程序让用户输入一组整数放入列表中，然后从列表中挑选出最大值和最小值。以 testLinuxBugListExtremum.py 为例，使用 pdb 调试。

下面先简单地介绍一下 pdb。pdb 在 Python 中是以模块的形式出现的，它是 Python 的标准库。可以在 Python 交互环境中使用，如图 2-28 所示。

```
king@debian:~/code/crawler$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pdb
>>> import testLinuxBugListExtremum
>>> pdb.run('testlinuxBugListExtremum.main')
> <string>(1)<module>()
(Pdb)
```

图 2-28　模块式使用 pdb

也可以在程序中间插入一段程序，相当于在一般 IDE 里面打上断点，然后启动 debug，不过这种方式是 hardcode 的，如图 2-29 所示。

```
def getMaxNum(List):
    #获取列表中最大值
    import pdb
    pdb.set_trace()
    num = List[0]
    for i in List[1:]:
        if num <= i:
            num = i
    return num

def getMinNum(List):
    #获取列表中最小值
    num = List[0]
    for i in List[1:]:
        if num >= i:
            num = i
    return num


if __name__ == '__main__':
    numList = getList()
    maxNum = getMaxNum(numList)
    print(u'列表中最大值为:%d' %maxNum)
                                                    50,19          93%
```

图 2-29　程序内使用 pdb

将 pdb 放入程序内，在运行程序时。运行到 pdb 行后就暂停了，然后开始运行 pdb 程序。这种方式需要改动程序，还是比较麻烦。

笔者更喜欢最后一种方法，命令行启动目标程序，加上-m 参数调用 pdb 模块，如图 2-30 所示。

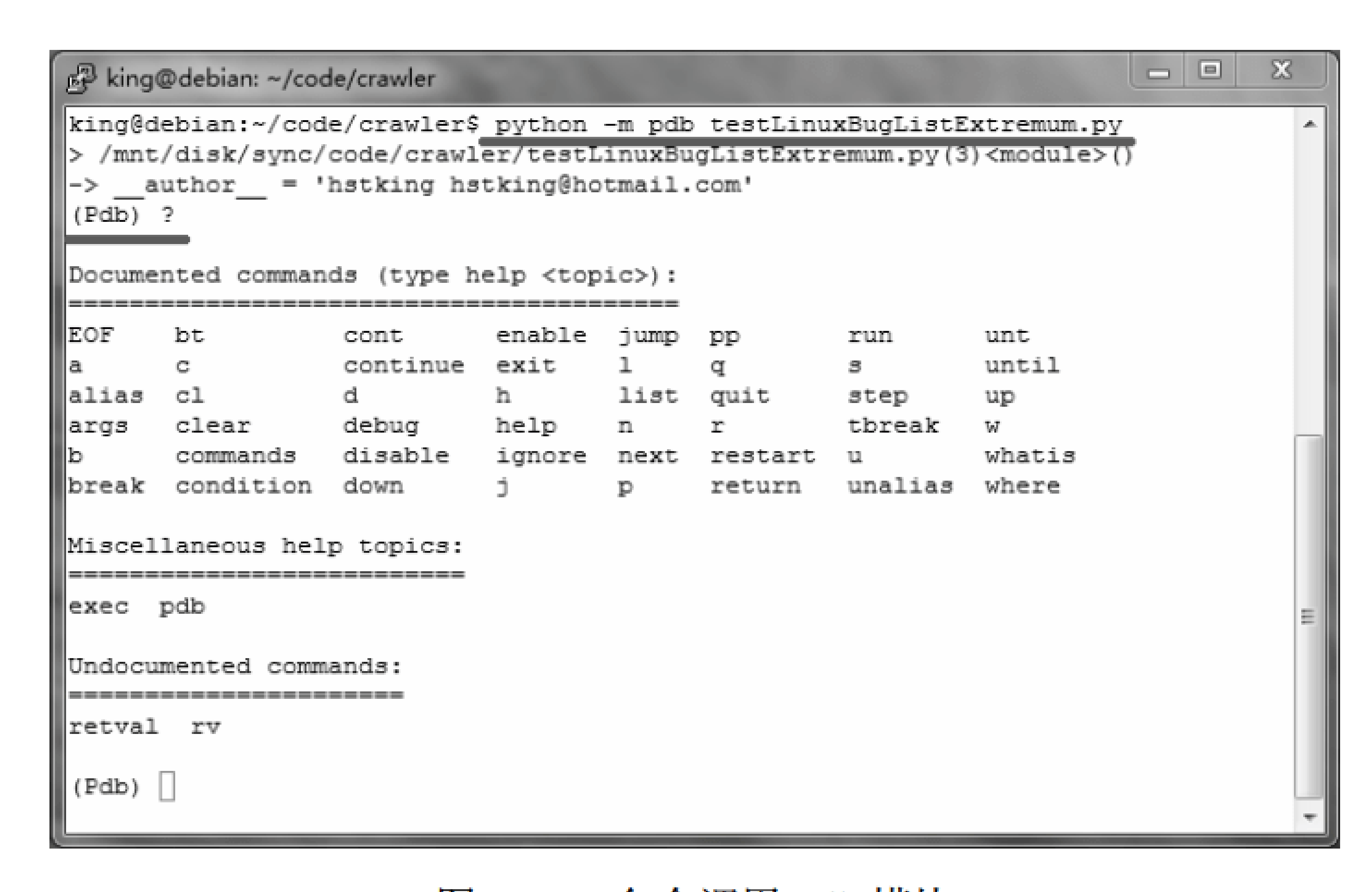


图 2-30　命令调用 pdb 模块

图 2-20 显示了 pdb 的所有命令，这里只说明最常用的几个：

- list：显示程序，可以带参数。比如显示第 5 行 list 5。
- break：添加断点。比如在第 5 行添加断点 break 5，在 getList 函数添加断点 break。
- run：开始运行程序。
- step：单步运行，进入函数内部。
- next：单步运行，不进入函数内部。
- print：显示参数。
- quit：退出 pdb。

下面开始调试 testLinuxBugListExtremum.py 程序。执行命令：

```
python -m pdb testLinuxBugListExtremum.py
list 52
break getList
break getMaxNum
break getMinNum
break
```

执行结果如图 2-31 所示。

```
king@debian: ~/code/crawler
king@debian:~/code/crawler$ python -m pdb testLinuxBugListExtremum.py
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(3)<module>()
-> __author__ = 'hstking hstking@hotmail.com'
(Pdb) list 52
 47         if __name__ == '__main__':
 48                 numList = getList()
 49                 maxNum = getMaxNum(numList)
 50                 print(u'列表中最大值为:%d' %maxNum)
 51                 minNum = getMinNum(numList)
 52                 print(u'列表中最小值为:%d' %minNum)
[EOF]
(Pdb) break getList
Breakpoint 1 at /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py:8
(Pdb) break getMaxNum
Breakpoint 2 at /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py:28
(Pdb) break getMinNum
Breakpoint 3 at /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py:38
(Pdb) break
Num Type         Disp Enb   Where
1   breakpoint   keep yes   at /mnt/disk/sync/code/crawler/testLinuxBugListExtre
mum.py:8
2   breakpoint   keep yes   at /mnt/disk/sync/code/crawler/testLinuxBugListExtre
mum.py:28
3   breakpoint   keep yes   at /mnt/disk/sync/code/crawler/testLinuxBugListExtre
mum.py:38
(Pdb)
```

图 2-31　pdb 加入断点

执行命令 run，开始运行程序，函数外的行使用 next 单步运行，到了函数入口后使用 step 单步运行。中途使用 print 命令随时监视变量变化，如图 2-32 所示。

```
king@debian: ~/code/crawler
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(15)getList()
-> print(u'结束构建列表，请按回车')
(Pdb)
结束构建列表，请按回车
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(16)getList()
-> num = raw_input('请输入一个整数: ')
(Pdb)
请输入一个整数: 10
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(17)getList()
-> if num == '':
(Pdb) print num
10
(Pdb) next
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(19)getList()
-> try:
(Pdb)
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(20)getList()
-> num = int(num)
(Pdb)
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(25)getList()
-> numList.append(num)
(Pdb)
> /mnt/disk/sync/code/crawler/testLinuxBugListExtremum.py(12)getList()
-> while num:
(Pdb) print numList
[10]
(Pdb)
```

图 2-32　调试 testLinuxDebugListExtremum.py

调试完毕后输入 quit，退出 pdb。pdb 没有 GUI，用起来似乎没有那么直观。用习惯了也还挺方便的。如果偏爱 GUI，那还是找个 Python IDE 吧，Eclipse + pydev 就很方便了。也是多平台通用，除了块头大一点，没什么缺点。

注意：pdb 是 Python 调试工具，它也是 Python 的标准模块之一，所以也可以用 import 将它导入到程序中使用。

2.6 本章小结

Python 的知识点远不止这一点，但读者了解了这些，又有一点其他编程语言的基础，基本就可以用 Python 来解决一些小问题了。如果需要继续深入，请自行参考教程或自行 Google。Python 是一门黏性非常强的语言，它可以调用别的语言来编写自己的模块，用来弥补自己的不足，因此也被称为胶水语言。虽然 Python 易学难精，但它是一个非常有用的编程语言，通用各大平台，值得投入精力深入学习。

第 3 章 简单的Python脚本

Python 的基础部分已经学完了，下一步可以开始写 Python 程序了。因为 Python 程序无须编译直接执行，所以也可以称之为脚本。在这里笔者把大一点的、复杂点的 Python 脚本称之为程序，把简单的 Python 程序称之为脚本。

3.1 九九乘法表

编写程序，由简到难。似乎没有比九九乘法表更简单的程序了吧，那就从九九乘法表开始。Python 的结构集合了 C 和 C++的优点，语法结构也相差不远，在编程时只需重点注意格式（其实就是空格）就可以了。

3.1.1 Project 分析

九九乘法表，从小学就开始学习，每个人都会背。如果把这个表格排列整齐一点就会发现它呈现出一个边长为 9 的直角三角形。这个图形从左到右横向是呈线性递加的。这样的话给出一个 for 循环正合适。而纵向是也有限（9 行）递加的，再给出一个 for 循环就可以了。

3.1.2 Project 实施

【示例 3-1】编写 table9x9.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi table9x9.py
```

table9x9.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

```

```

class PrintTable(object):
    '''打印九九乘法表 '''
    def __init__(self):
        print(u'开始打印 9x9 的乘法表格')
        self.print99()

    def print99(self):
        for i in xrange(1,10):
            for j in xrange(1,i+1):
                print('%dX%d=%2s  ' %(j,i,i*j)),
            print('\n')


if __name__ == '__main__':
    pt = PrintTable()
```

输入:wq，保存 table9x9.py。table9x9.py 用于打印一个九九乘法表格。执行命令：

```
python table9x9.py
```

得到的结果如图 3-1 所示。

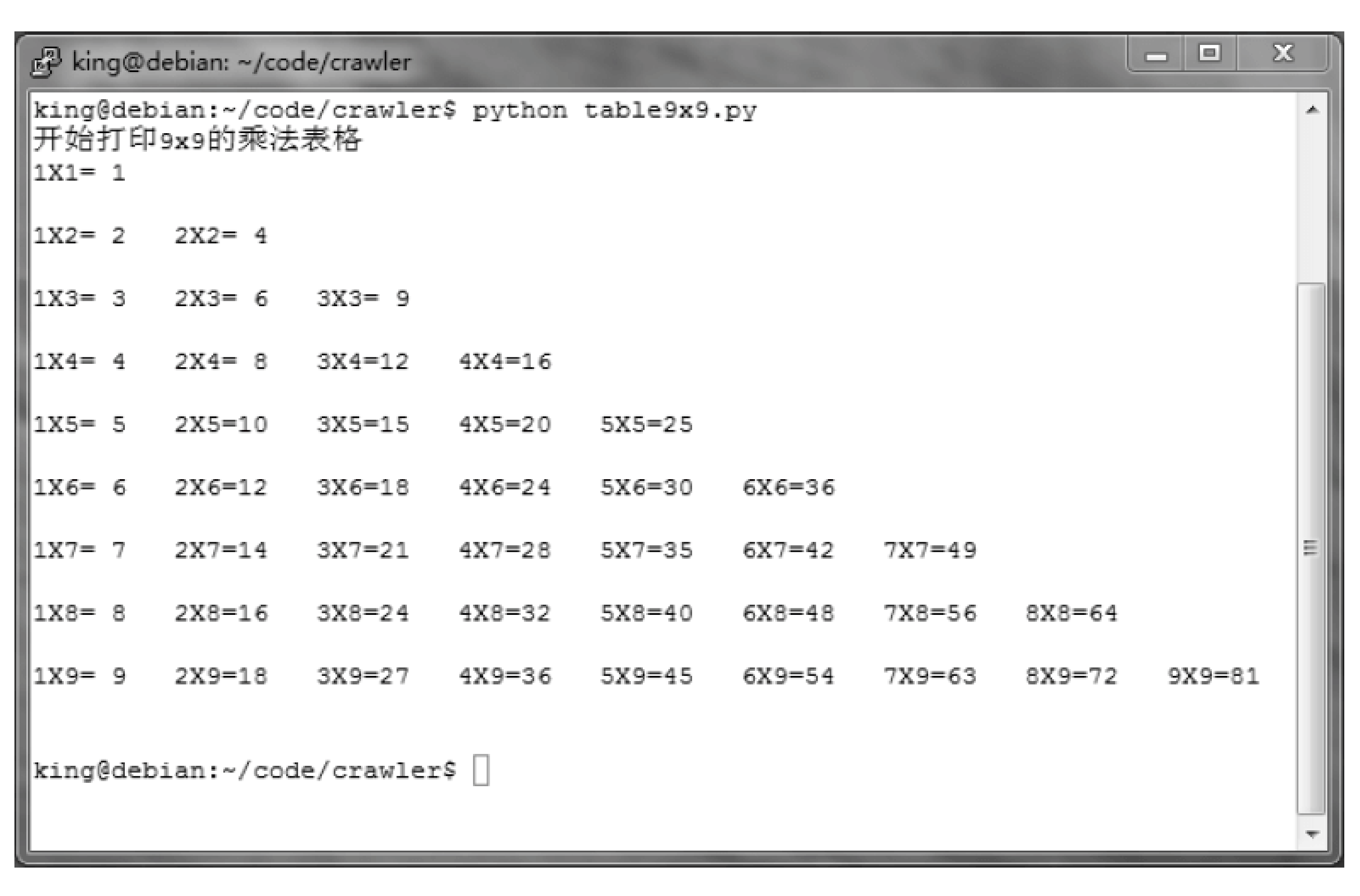


图 3-1　乘法表

十几行的代码，如果愿意精简，甚至可以把代码压缩到十行以内。足够简单了吧。

3.2 斐波那契数列

斐波那契数列（Fibonacci sequence），又称黄金分割数列，因数学家列昂纳多·斐波那契（Leonardoda Fibonacci ）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34……在数学上，斐波那契数列以如下被以递归的方法定义：F（0）=0、F（1）=1、F（n）=F(n-1)+F(n-2)（n≥2，n∈N*）。

3.2.1 Project 分析

从斐波那契数列的定义上可以看出，求斐波那契数列最正统的方法就是函数递归了。不过，对于 Python 而言，有更加简单的方法操作。这得益于 Python 独有的数据类型——列表。Python 列表可以使用 append 方法在列表的尾部追加数据。这样一来，求斐波那契数列就成了简单的加法游戏，无须递归求解了（可惜 C 语言中没有变长数列，否则在 C 语言中求斐波那契数列也简单了）。

3.2.2 Project 实施

【示例 3-2】编写 fibonacci.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi fibonacci.py
```

fibonacci.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


class Fibonacci(object):
    '''返回一个 fibonacci 数列 '''
    def __init__(self):
        self.fList = [0,1] #设置初始列表
        self.main()

    def main(self):
        listLen = raw_input('请输入 fibonacci 数列的长度(3-50):')
        self.checkLen(listLen)
        while len(self.fList) < int(listLen):
            self.fList.append(self.fList[-1] + self.fList[-2])
        print('得到的 fibonacci 数列为:\n %s ' %self.fList)

```

```
    def checkLen(self,lenth):  #检查输入的数据是否符合要求
        lenList = map(str,xrange(3,51))
        if lenth in lenList:
            print(u'输入的长度符合标准，继续运行')
        else:
            print(u'只能输入 3-50，太长了不是算不出，只是没必要')
            exit()


if __name__ == '__main__':
    f = Fibonacci()
```

输入:wq，保存 fibonacci.py。fibonacci.py 用于创建一个定长列表，该列表就是斐波那契数列。执行命令：

```
python fibonacci.py
```

得到的结果如图 3-2 所示。

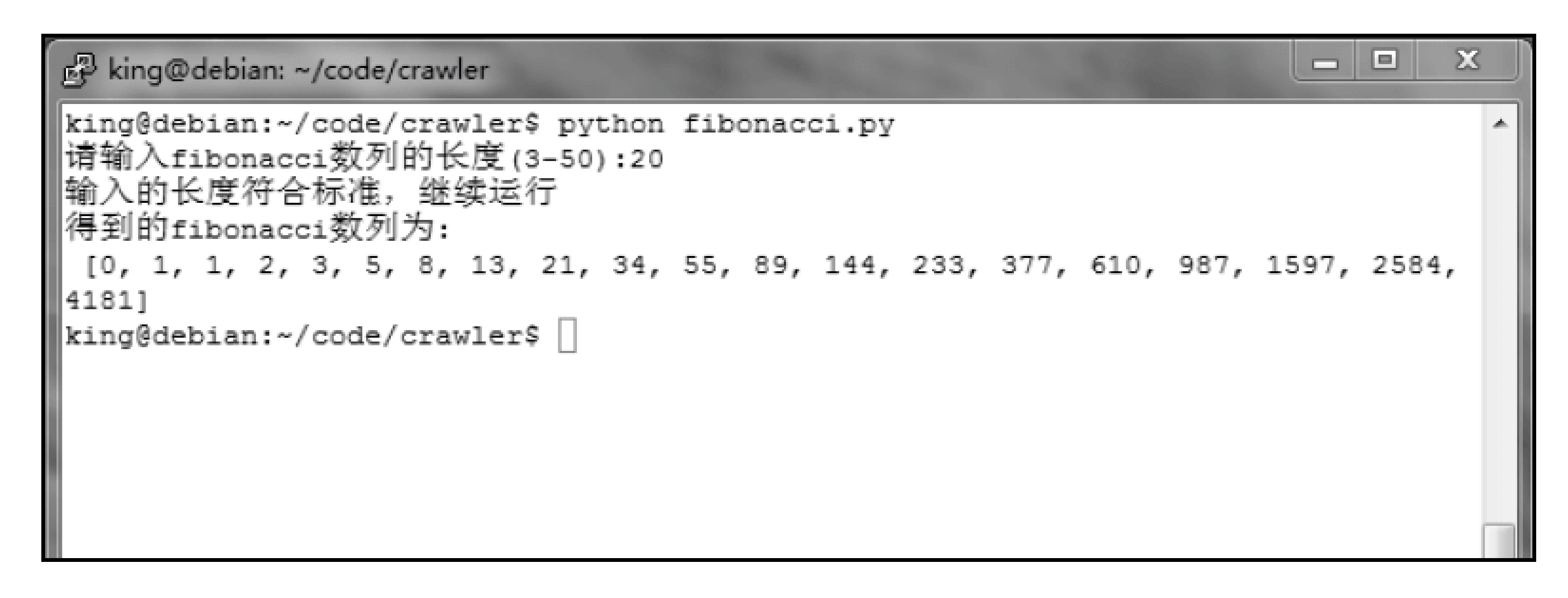


图 3-2　fibonacci 数列

Python 有独特的列表类型，在获取递归队列时有独特的优势。

3.3 概率计算

将理想状态绝对无误差的 10 个同样的小球从 1~10 标号，然后随机从中选出 1 个小球。如果选取的次数足够多，就可以计算各个小球被选取出来的概率。编写一个 Python 程序来算一算，看看老天偏爱哪个数。

3.3.1　Project 分析

这是一个随机数的问题。Python 有个 random 模块，专门用来解决这类问题。据说 Python 用 random 选取出来的随机数都是伪随机数。不过也没关系，只需要算出大致的结果就可以了。没计算之前，个人认为每个球被选取出来的概率都一样。下面就来算算看。

3.3.2 Project 实施

【示例 3-3】编写 ball.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi ball.py
```

ball.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import random

class SelectBall(object):
    def __init__(self):
        self.run()

    def run(self):
        while True:
            numStr = raw_input('输入测试的次数：')
            try:
                num = int(numStr)
            except ValueError:
                print(u'要求输入一个整数')
                continue
            else:
                break
        ball = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        for i in xrange(num):
            n = random.randint(1,10)
            ball[n-1] += 1
        for i in xrange(1,11):
            print(u'获取第%d 号球的概率为%f' %(i, ball[i-1]*1.0/num))


if __name__ == '__main__':
    SB = SelectBall()
```

执行命令：

```
python ball.py
```

得到的结果如图 3-3 所示。

```
king@debian: ~/code/crawler
输入测试的次数：10000
获取第1号球的概率为0.100200
获取第2号球的概率为0.102300
获取第3号球的概率为0.097200
获取第4号球的概率为0.099400
获取第5号球的概率为0.101900
获取第6号球的概率为0.097600
获取第7号球的概率为0.101000
获取第8号球的概率为0.102400
获取第9号球的概率为0.096600
获取第10号球的概率为0.101400
king@debian:~/code/crawler$ python ball.py
输入测试的次数：1000000
获取第1号球的概率为0.099808
获取第2号球的概率为0.099832
获取第3号球的概率为0.100118
获取第4号球的概率为0.099540
获取第5号球的概率为0.099962
获取第6号球的概率为0.099900
获取第7号球的概率为0.100511
获取第8号球的概率为0.100661
获取第9号球的概率为0.099647
获取第10号球的概率为0.100021
king@debian:~/code/crawler$
```

图 3-3　选球概率

果然如此，每个球选取的概率差不多。选取的次数越多，这个趋势就越明显。那就是说，在理想状态下，所有球被选取的概率是一样的。



这种选取小球概率的计算方法只是一种理想状态的算法。类似于丢硬币出现正反面的概率，理论上应该是一半对一半，但实际上由于硬币材质的缘故，丢硬币的次数越多，正反面出现的概率差距就越大。

3.4 读写文件

读写文件是最常见的 IO 操作。Python 内置了读写文件的函数，用法和 C 是兼容的。在磁盘上读写文件的功能都是由操作系统提供的，现代操作系统不允许普通的程序直接操作磁盘，所以，读写文件就是请求操作系统打开一个文件对象（通常称为文件描述符），然后，通过操作系统提供的接口从这个文件对象中读取数据（读文件），或者把数据写入这个文件对象（写文件）。

3.4.1 Project 分析

Python 使用内置函数 open 来读写文件。查看 open 函数的帮助文档。执行命令：

```
python
help(open)
```

执行的结果如图 3-4 所示。

```
Help on built-in function open in module __builtin__:

open(...)
    open(name[, mode[, buffering]]) -> file object

    Open a file using the file() type, returns a file object.  This is the
    preferred way to open a file.  See file.__doc__ for further information.
(END)
```

图 3-4　help open

图 3-4 中的 Name 是需要操作的文件名，mode 是模式。这个模式共有 7 种，如表 3-1 所示。

表 3-1　Python Open Mode

r	以读方式打开文件，可读取文件信息
w	以写方式打开文件，可向文件写入信息。如文件存在，则清空该文件，再写入新内容
a	以追加模式打开文件，如果文件不存在，则创建
r+	以读写方式打开文件，可对文件进行读和写操作
w+	消除文件内容，然后以读写方式打开文件
a+	以读写方式打开文件，并把文件指针移到文件尾
b	以二进制模式打开文件，而不是以文本模式

这 7 中模式可以组合使用。下面将用 Python 创建一个文件，并写入、读取内容。

3.4.2　project 实施

【示例 3-4】编写 operaFile.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi operaFile.py
```

operaFile.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import os

def operaFile():  #创建文件
    print(u'创建一个名字为 test.txt 的文件，并在其中写入 Hello Python')
    print(u'先得保证 test.txt 不存在')
    os.system('rm test.txt')
    os.system('ls -l test.txt')
    print(u'现在再来创建文件并写入内容\n')
    fp = open('test.txt', 'w')
```

```
    fp.write('Hello Python')
    fp.close()
    print(u'不要忘记用 close 关闭文件哦')
    print(u'再来看看 test.txt 是否存在，和内容\n')
    os.system('ls -l test.txt')
    os.system('cat test.txt')
    print('\n')

    print(u'如何避免 open 文件失败的问题呢？')
    print(u'使用 with as 就可以了')
    with open('test.txt', 'r') as fp:
        st = fp.read()
    print('test.txt 的内容为:%s' %st)

if __name__ == '__main__':
    operaFile()
```

执行命令：

```
python operaFile.py
```

得到的结果如图 3-5 所示。

```
king@debian:~/code/crawler$
king@debian:~/code/crawler$ python operaFile.py
创建一个名字为test.txt的文件，并在其中写入Hello Python
先得保证test.txt不存在
ls: 无法访问test.txt: 没有那个文件或目录
现在再来创建文件并写入内容

不要忘记用close关闭文件哦
再来看看test.txt是否存在，和内容

-rw-r--r-- 1 king king 12 9月  11 21:40 test.txt
Hello Python

如何避免open文件失败的问题呢？
使用with as就可以了
test.txt的内容为:Hello Python
king@debian:~/code/crawler$
```

图 3-5　Python 读写文件

Python 对文件的操作跟 C 很类似，但功能远比 C 要丰富。例如按行读取文件，多行读取文件等等。C 语言的优势是快，Python 的优势是模块丰富。

3.5 本章小结

本章的几个 Python 小程序都比较简单。程序简单没关系，只要可以解决问题就行。学习 Python 最快的方法就是多写程序，用程序解决实际问题。Python 并不复杂，多写、多做、多练很快就能掌握。

第 4 章

◂Python爬虫常用模块▸

Python 最强大的方面就体现在它那近乎无限的模块库上。相信没有人能熟悉所有的模块功能，也没这个必要。只需要了解标准模块库就可以解决大部分的问题了，特殊需求先找第三方的模块。如果还是解决不了问题，那就到 github 碰碰运气。如果实在是运气不佳，那就自己动手丰衣足食吧。Python 2.7 标准模块库的官方文档可参考 https://docs.python.org/2.7/py-modindex.html。本章只讲解与网络爬虫有关的常用模块。

4.1 Python 标准库之 urllib2 模块

涉及网络这块，必不可少的模块就是 urllib2 了。顾名思义这个模块主要负责打开 URL 和 HTTP 协议之类的，还有一个模块叫 urllib，但它们并不是升级版的关系。具体可见 google 文章 Python: difference between urllib and urllib2，讲得很透彻，这里就不重复了。urllib2 模块的官方文档可参考 https://docs.python.org/2.7/library/urllib2.html#module-urllib2。

4.1.1 urllib2 请求返回网页

urllib2 最简单的应用就是 urllie2.urlopen 了，函数使用如下：

```
   urllib2.urlopen(url[, data[, timeout[, cafile[, capath[, cadefault[,
context]]]]]]
```

按照官方文档，urllib2.urlopen 可以打开 HTTP、HTTPS、FTP 协议的 URL。主要应用于 HTTP 协议。它参数中以 ca 开头的都是跟身份验证有关，不太常用。data 参数是以 post 方式提交 URL 时使用的。最常用就只有 URL 和 timeout 参数了。url 参数是提交的网络地址（地址全称，前端需协议名，后端需端口，比如 http://192.168.1.1:80），timeout 是超时时间设置。

函数返回对象有 3 个额外的使用方法。geturl()函数返回 response 的 url 信息，常用于 url 重定向的情况。info()函数返回 response 的基本信息。getcode()函数返回 response 的状态代码，最常见的代码是 200 服务器成功返回网页，404 请求的网页不存在，503 服务器暂时不可用。

【示例 4-1】测试使用 urllib2 模块打开百度的首页。编写 testUrllib2.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testUrllib2.py
```

testUrllib2.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import urllib2

def clear():
    '''该函数用于清屏 '''
    print(u'内容较多，显示 3 秒后翻页')
    time.sleep(3)
    OS = platform.system()
    if (OS == u'Windows'):
        os.system('cls')
    else:
        os.system('clear')

def linkBaidu():
    url = 'http://www.baidu.com'
    try:
        response = urllib2.urlopen(url,timeout=3)
    except urllib2.URLError:
        print(u"网络地址错误")
        exit()
    with open('./baidu.txt','w') as fp:
        fp.write(response.read())
    print(u"获取 url 信息，response.geturl() \n: %s" %response.geturl())
    print(u"获取返回代码，response.getcode() \n: %s" %response.getcode())
    print(u"获取返回信息，response.info() \n: %s" %response.info())
    print(u"获取的网页内容已存入当前目录的 baidu.txt 中，请自行查看")


if __name__ == '__main__':
    linkBaidu()
```

输入:wq，保存 testUrllib2.py。testUrllib2.py 调用 urllib2 模块请求百度的主页，显示返回的信息并将服务器答复的数据保存到 baidu.txt 中以备查询。执行命令：

```
python testUrllib2.py
```

得到的结果如图 4-1 所示。

```
king@debian:~/code/crawler$ python testUrllib2.py
获取url信息, response.geturl()
: http://www.baidu.com
获取返回代码, response.getcode()
: 200
获取返回信息, response.info()
: Date: Thu, 30 Jun 2016 04:41:00 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: Close
Vary: Accept-Encoding
Set-Cookie: BAIDUID=D411AF0BE6CF9C97BFE      D707E3BD:FG=1; expires=Thu, 31-Dec-3
7 23:55:55 GMT; max-age=2147483647; pat      iomain=.baidu.com
Set-Cookie: BIDUPSID=D411AF0BE6CF9C97BF      BD707E3BD; expires=Thu, 31-Dec-37 23
:55:55 GMT; max-age=2147483647; path=/;      ain=.baidu.com
Set-Cookie: PSTM=1467261660; expires=T      1-Dec-37 23:55:55 GMT; max-age=214748
3647; path=/; domain=.baidu.com
Set-Cookie: BDSVRTM=0; path=/
Set-Cookie: BD_HOME=0; path=/
Set-Cookie: H_PS_PSSID=1461_205       _20388_20417_17001_15297_12370; path=/; d
omain=.baidu.com
P3P: CP=" OTI DSP COR IVA OU      "
Cache-Control: private
Cxy_all: baidu+ffe990241      3d7816cf3df7f8
Expires: Thu, 30 Jun 2      .21 GMT
X-Powered-By: HPHP
Server: BWS/1.1
X-UA-Compatible:       ,chrome=1
BDPAGETYPE: 1
BDQID: 0xc1b      0001c523
BDUSERID: 0

获取的网页内容已存入当前目录的baidu.txt中, 请自行查看
king@debian:~/code/crawler$
```

图 4-1　run testUrllib2.py

从 baidu.txt 的结果可以看出百度首页的页面虽然很简洁，但内容还是很丰富的。最简单的 urllib2 用法就是这样了。至于那些跟身份验证有关的参数，如有需要请自行参考官方文档或 google。

4.1.2　urllib2 使用代理访问网页

在使用网络爬虫时，有的网站拒绝了一些 IP 的直接访问。这时就不得不利用代理了。urllib2 添加代理的方式不止一种。这里选择最简单也是最直接的一种为例，至于免费的代理，网络上很多，可自行搜索一下，选择一个确定可用的 Proxy，这里笔者选择的是从 www.youdaili.net 中获取的代理 36.7.172.18:82@HTTP。

【示例 4-2】编写 testUrllib2WithProxy.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testUrllib2WithProxy.py
```

testUrllib2WithProxy.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
```

```
__author__ = 'hstking hstking@hotmail.com'

import urllib2
import sys
import re

def testArgument():
    '''测试输入参数，只需要一个参数 '''
    if len(sys.argv) != 2:
        print(u'只需要一个参数就够了')
        tipUse()
        exit()
    else:
        TP = TestProxy(sys.argv[1])

def tipUse():
    '''显示提示信息 '''
    print(u'该程序只能输入一个参数，这个参数必须是一个可用的 proxy')
    print(u'usage: python testUrllib2WithProxy.py http://1.2.3.4:5')
    print(u'usage: python testUrllib2WithProxy.py https://1.2.3.4:5')


class TestProxy(object):
    '''这个类的作用是测试 proxy 是否有效 '''
    def __init__(self,proxy):
        self.proxy = proxy
        self.checkProxyFormat(self.proxy)
        self.url = 'http://www.baidu.com'
        self.timeout = 5
        self.flagWord = '百度' #在网页返回的数据中查找这个关键词
        self.useProxy(self.proxy)

    def checkProxyFormat(self,proxy):
        try:
            proxyMatch =
re.compile('http[s]?://[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}:[\d]{1,5}$')
            re.search(proxyMatch,proxy).group()
        except AttributeError:
            tipUse()
            exit()
        flag = 1
        proxy = proxy.replace('//','')
        try:
```

```
            protocol = proxy.split(':')[0]
            ip = proxy.split(':')[1]
            port = proxy.split(':')[2]
        except IndexError:
            print(u'下标出界')
            tipUse()
            exit()
        flag = flag and len(proxy.split(':')) == 3 and len(ip.split('.'))
== 4
        flag = ip.split('.')[0] in map(str,xrange(1,256)) and flag
        flag = ip.split('.')[1] in map(str,xrange(256)) and flag
        flag = ip.split('.')[2] in map(str,xrange(256)) and flag
        flag = ip.split('.')[3] in map(str,xrange(1,255)) and flag
        flag = protocol in [u'http',u'https'] and flag
        flag = port in map(str,range(1,65535)) and flag
        '''这里是在检查proxy的格式 '''
        if flag:
            print(u'输入的http代理服务器符合标准')
        else:
            tipUse()
            exit()

    def useProxy(self,proxy):
        '''利用代理访问百度，并查找关键词 '''
        protocol = proxy.split('//')[0].replace(':','')
        ip = proxy.split('//')[1]
        opener = urllib2.build_opener(urllib2.ProxyHandler({protocol:ip}))
        urllib2.install_opener(opener)
        try:
            response = urllib2.urlopen(self.url,timeout = self.timeout)
        except:
            print(u'连接错误，退出程序')
            exit()
        str = response.read()
        if re.search(self.flagWord,str):
            print(u'已取得特征词，该代理可用')
        else:
            print(u'该代理不可用')


if __name__ == '__main__':
    testArgument()
```

输入:wq，保存 testUrllib2WithProxy.py。testUrllib2WithProxy.py 将使用代理来访问百度

的首页，并设定了一个特征词。如果能在返回的结果中取得这个特征词就说明该 proxy 是可用的。执行命令：

```
python testUrllib2WithProxy.py http://36.7.172.18:82
```

得到的结果如图 4-2 所示。

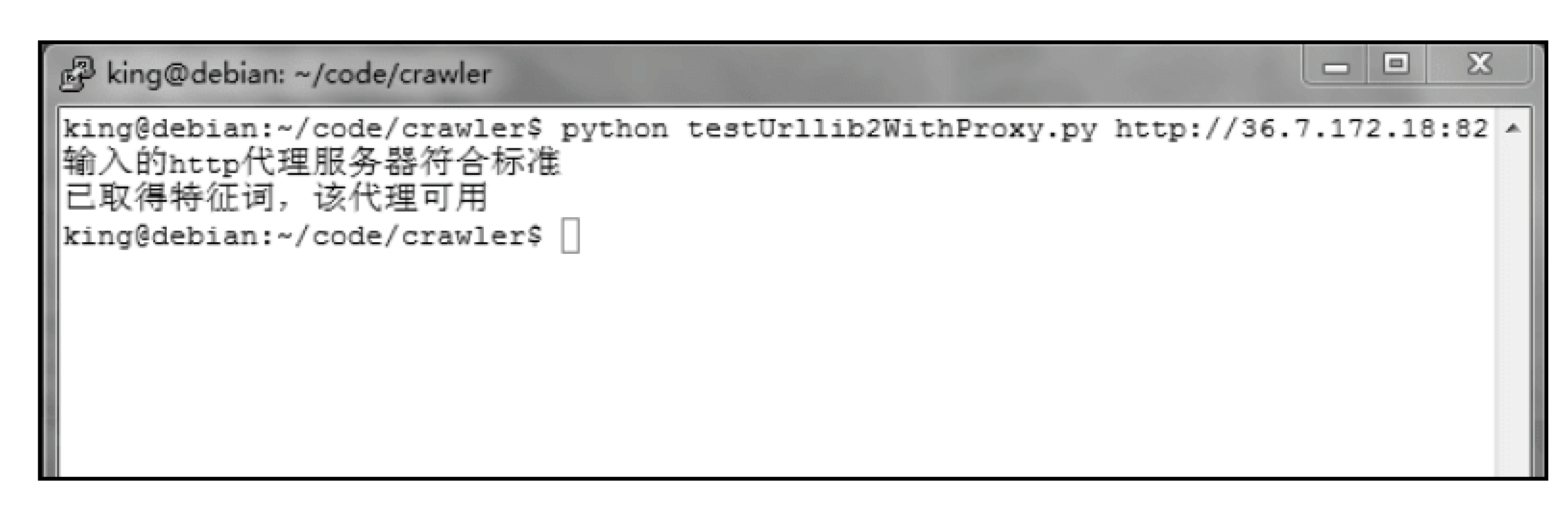


图 4-2 run testUrllib2WithProxy.py

至此，urllib2 使用代理打开网页测试完毕。这个程序采用的是函数与类混合的形式。这个程序无须修改即可作为模块导入到其他程序中用于测试 Proxy 是否可用。



urllib2 是 Python 中使用率非常高的模块。有很多第三方模块都是通过包装这个模块的功能而开发出来的。这是个必须掌握的模块。

4.1.3 urllib2 修改 header

在使用网络爬虫时，有一些站点不喜欢被程序（非人为访问）访问，会检查连接者的“身份证”。默认情况下 urllib2 把自己的版本号 Python-urllib/x.y（x 和 y 是 Python 主版本和次版本号，例如 Python-urllib/2.7）作为“身份证号码”来通过检查。这个“身份证号码”可能会让站点迷惑，或者干脆不工作。这时可以让 Python 程序冒充浏览器访问网站。网站是通过浏览器发送过来的 User-Agent 的值来确认浏览器身份的。用 urllib2 创建一个请求对象，并给它一个包含头数据的字典，修改 User-Agent 欺骗网站。一般来说，把 User-Agent 修改成 Internet Explorer 是最安全的，改成其他的也行。

先将准备工作做好，将所有常见的 User-Agent 全部放到一个 userAgents.py 文件中，以字典的形式保存起来，方便以后当成模块导入使用。userAgents.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

pcUserAgent = {
"safari 5.1 - MAC":"User-Agent:Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_8; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50",
"safari 5.1 - Windows":"User-Agent:Mozilla/5.0 (Windows; U; Windows NT 6.1; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50",
```

```
"IE 9.0":"User-Agent:Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0;",
"IE 8.0":"User-Agent:Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0)",
"IE 7.0":"User-Agent:Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)",
"IE 6.0":"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
"Firefox 4.0.1 - MAC":"User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:2.0.1) Gecko/20100101 Firefox/4.0.1",
"Firefox 4.0.1 - Windows":"User-Agent:Mozilla/5.0 (Windows NT 6.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1",
"Opera 11.11 - MAC":"User-Agent:Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; en) Presto/2.8.131 Version/11.11",
"Opera 11.11 - Windows":"User-Agent:Opera/9.80 (Windows NT 6.1; U; en) Presto/2.8.131 Version/11.11",
"Chrome 17.0 - MAC":"User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_0) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11",
"Maxthon":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Maxthon 2.0)",
"Tencent TT":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; TencentTraveler 4.0)",
"The World 2.x":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)",
"The World 3.x":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; The World)",
"sogou 1.x":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; SE 2.X MetaSr 1.0; SE 2.X MetaSr 1.0; .NET CLR 2.0.50727; SE 2.X MetaSr 1.0)",
"360":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; 360SE)",
"Avant":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Avant Browser)",
"Green Browser":"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)"
}

mobileUserAgent = {
"iOS 4.33 - iPhone":"User-Agent:Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8J2 Safari/6533.18.5",
"iOS 4.33 - iPod Touch":"User-Agent:Mozilla/5.0 (iPod; U; CPU iPhone OS
```

```
4_3_3 like Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko)
Version/5.0.2 Mobile/8J2 Safari/6533.18.5",
    "iOS 4.33 – iPad":"User-Agent:Mozilla/5.0 (iPad; U; CPU OS 4_3_3 like
Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2
Mobile/8J2 Safari/6533.18.5",
    "Android N1":"User-Agent: Mozilla/5.0 (Linux; U; Android 2.3.7; en-us;
Nexus One Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1",
    "Android QQ":"User-Agent: MQQBrowser/26 Mozilla/5.0 (Linux; U; Android
2.3.7; zh-cn; MB200 Build/GRJ22; CyanogenMod-7) AppleWebKit/533.1 (KHTML, like
Gecko) Version/4.0 Mobile Safari/533.1",
    "Android Opera ":"User-Agent: Opera/9.80 (Android 2.3.4; Linux; Opera
Mobi/build-1107180945; U; en-GB) Presto/2.8.149 Version/11.10",
    "Android Pad Moto Xoom":"User-Agent: Mozilla/5.0 (Linux; U; Android 3.0;
en-us; Xoom Build/HRI39) AppleWebKit/534.13 (KHTML, like Gecko) Version/4.0
Safari/534.13",
    "BlackBerry":"User-Agent: Mozilla/5.0 (BlackBerry; U; BlackBerry 9800;
en) AppleWebKit/534.1+ (KHTML, like Gecko) Version/6.0.0.337 Mobile
Safari/534.1+",
    "WebOS HP Touchpad":"User-Agent: Mozilla/5.0 (hp-tablet; Linux;
hpwOS/3.0.0; U; en-US) AppleWebKit/534.6 (KHTML, like Gecko) wOSBrowser/233.70
Safari/534.6 TouchPad/1.0",
    "Nokia N97":"User-Agent: Mozilla/5.0 (SymbianOS/9.4; Series60/5.0
NokiaN97-1/20.0.019; Profile/MIDP-2.1 Configuration/CLDC-1.1) AppleWebKit/525
(KHTML, like Gecko) BrowserNG/7.1.18124",
    "Windows Phone Mango":"User-Agent: Mozilla/5.0 (compatible; MSIE 9.0;
Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0; HTC; Titan)",
    "UC":"User-Agent: UCWEB7.0.2.37/28/999",
    "UC standard":"User-Agent: NOKIA5700/ UCWEB7.0.2.37/28/999",
    "UCOpenwave":"User-Agent: Openwave/ UCWEB7.0.2.37/28/999",
    "UC Opera":"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; )
Opera/UCWEB7.0.2.37/28/999"
}
```

userAgents.py 里只包含了最常用的 User-Agent，如有特殊需要可以继续添加。但在使用网络爬虫时最好用最常见的那些 User-Agent，以免被网站拒绝访问。

【示例 4-3】准备工作完毕后，开始编写 testUrllib2ModifyHeader.py。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testUrllib2ModifyHeader.py
```

testUrllib2ModifyHeader.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import urllib2
import userAgents
'''userAgents.py 是个自定义的模块，位置处于当前目录下 '''

class Urllib2ModifyHeader(object):
    '''使用 urllib2 模块修改 header '''
    def __init__(self):
        #这个是 PC + IE 的 User-Agent
        PIUA = userAgents.pcUserAgent.get('IE 9.0')
        #这个是 Mobile + UC 的 User-Agent
        MUUA = userAgents.mobileUserAgent.get('UC standard')
        #测试用的网站选择的是有道翻译
        self.url = 'http://fanyi.youdao.com'

        self.useUserAgent(PIUA,1)
        self.useUserAgent(MUUA,2)

    def useUserAgent(self,userAgent,name):
        request = urllib2.Request(self.url)
request.add_header(userAgent.split(':')[0],userAgent.split(':')[1])
        response = urllib2.urlopen(request)
        fileName = str(name) + '.html'
        with open(fileName,'a') as fp:
            fp.write("%s\n\n" %userAgent)
            fp.write(response.read())

if __name__ == '__main__':
    umh = Urllib2ModifyHeader()
```

输入:wq，保存 testUrllib2ModifyHeader.py。testUrllib2ModifyHeader.py 使用了 2 个不同的浏览器 header 访问有道翻译的主页，并将返回的结果保存起来。执行命令：

```
python testUrllib2ModifyHeader.py
```

得到了 1.html 和 2.html，打开这两个网页比较一下，得到的结果如图 4-3 所示。



图 4-3　run testUrllib2ModifyHeader.py

urllib2 添加 Header 打开网页测试完毕。同一网站会给不同的浏览器返回不同的内容。所以在使用网络爬虫时，如果条件允许尽可能添加一个固定的 User-Agent。

4.2　Python 标准库——logging 模块

Logging 模块，顾名思义就是针对日志的。到目前为止，所有的程序标准输出（输出到屏幕）都是使用的 print 函数。Logging 模块可以替代 print 函数的功能，并能将标准输出输入到日志文件保存起来，而且利用 logging 模块可以部分替代 debug 的功能，给程序排错。

4.2.1　简述 logging 模块

首先要说到的是 logging 模块的几个级别。默认情况下 logging 模块有 6 个级别。它们分别是 NOTSET 值为 0、DEBUG 值为 10、INFO 值为 20、WARNING 值为 30、ERROR 值为 40、CRITICAL 值为 50（也可以自定义级别）。这些级别的用处是，先将自己的日志定一个级别，logging 模块发出的信息级别高于定义的级别，将在标准输出（屏幕）显示出来。发出的信息级别低于定义的级别则略过。如果未定义级别，默认定义的级别是 WARNING。

先测试一下。在 Windows 中打开 IDLE，执行命令：

```
import logging
```

```
logging.NOTSET
logging.DEBUG
logging.INFO
logging.WARNING
logging.ERROR
logging.CRITICAL
logging.debug("debug message")
logging.info("info message")
logging.warning("warning message")
logging.error("error message")
logging.critical("critical message")
```

执行结果如图 4-4 所示。

```
Python 2.7.11 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500 64 bit (
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import logging
>>> logging.NOTSET
0
>>> logging.DEBUG
10
>>> logging.INFO
20
>>> logging.WARNING
30
>>> logging.ERROR
40
>>> logging.CRITICAL
50
>>> logging.debug('debug message')
>>> logging.info('info message')
>>> logging.warning('warning message')
WARNING:root:warning message
>>> logging.error('error message')
ERROR:root:error message
>>> logging.critical('critical message')
CRITICAL:root:critical message
>>>
```

图 4-4　logging 级别

使用 logging 最简单的方法就是 logging.basicConfig。logging.basicConfig 的应用方法为：

```
logging.basicConfig([**kwargs])
```

这个函数可用的参数有：

- filename：用指定的文件名创建 FiledHandler（后边会具体讲解 handler 的概念），这样日志会被存储在指定的文件中。
- filemode：文件打开方式，在指定了 filename 时使用这个参数，默认值为“a”还可指定为“w”。
- format：指定 handler 使用的日志显示格式。
- datefmt：指定日期时间格式。
- level：设置 rootlogger（后边会讲解具体概念）的日志级别。
- stream：用指定的 stream 创建 StreamHandler。可以指定输出到 sys.stderr,sys.stdout

或者文件，默认为 sys.stderr。若同时列出了 filename 和 stream 两个参数，则 stream 参数会被忽略。

参数中的 format 参数可能用到的格式化串：

- %(name)s：Logger 的名字。
- %(levelno)s：数字形式的日志级别。
- %(levelname)s：文本形式的日志级别。
- %(pathname)s：调用日志输出函数的模块的完整路径名，可能没有。
- %(filename)s：调用日志输出函数的模块的文件名。
- %(module)s：调用日志输出函数的模块名。
- %(funcName)s：调用日志输出函数的函数名。
- %(lineno)d：调用日志输出函数的语句所在的代码行。
- %(created)f：当前时间，用 UNIX 标准的表示时间的浮点数表示。
- %(relativeCreated)d：输出日志信息时，自 Logger 创建以来的毫秒数。
- %(asctime)s：字符串形式的当前时间。默认格式是 "2003-07-08 16:49:45,896"。逗号后面的是毫秒。
- %(thread)d：线程 ID。可能没有。
- %(threadName)s：线程名。可能没有。
- %(process)d：进程 ID。可能没有。
- %(message)s：用户输出的消息。

还有一些参数应用与进程线程等高级应用，可自行参考官方文档或 Google。

参数中的 datefmt 是日期的格式化，最常用的几个格式化是：

- %Y：年份的长格式，如 1999。
- %y：年份的短格式，如 99。
- %m：月份，01~12。
- %d：日期，01~31。
- %H：小时，0~23。
- %w：星期，0~6，星期天是 0。
- %M：分钟，00~59。
- %S：秒，00~59。

日期格式化的参数还有一些不太常用的，这里就不多做介绍了。读者可自行参考官方文档或 Google。

【示例 4-4】下面利用 logging.basicConfig 写个最基本的日志模块应用程序中，编写 testLogging.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testLogging.py
```

```
cat testLog.txt
```

testLogging.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import logging

class TestLogging(object):
    def __init__(self):
        logFormat = '%(asctime)-12s %(levelname)-8s %(name)-10s
%(message)-12s'
        logFileName = './testLog.txt'

        logging.basicConfig(level = logging.INFO,
format = logFormat,
filename = logFileName,
filemode = 'w')

        logging.debug('debug message')
        logging.info('info message')
        logging.warning('warning message')
        logging.error('error message')
        logging.critical('critical message')


if __name__ == '__main__':
    tl = TestLogging()
```

输入:wq，保存 testLogging.py。testlogging.py 用于测试 logging 模块。该脚本使用不同的级别向 logger 发送了几条信息。执行命令：

```
python testLogging.py
cat testLog.txt
```

得到的结果如图 4-5 所示。

```
king@debian: ~/code/crawler
king@debian:~/code/crawler$ python testLogging.py
king@debian:~/code/crawler$ cat testLog.txt
2016-06-30 14:26:31,172 INFO     root       info message
2016-06-30 14:26:31,173 WARNING  root       warning message
2016-06-30 14:26:31,173 ERROR    root       error message
2016-06-30 14:26:31,173 CRITICAL root       critical message
king@debian:~/code/crawler$
```

图 4-5　run testLogging.py

默认的 logging 级别是 logging.INFO（程序第 12 行）。而 logging.debug（程序第 17 行）的级别低于 logging.INFO，所以没有显示。

在程序中的关键位置插入 log 信息，执行 python 程序时出现什么问题，可以直接查找日志文件，无须再一步步地 debug 调试。

4.2.2　自定义模块 myLog

使用 logging 模块很方便，但在编写过程中添加一大堆的代码就不是那么愉快的事情了。好在 Python 有强大的 import，完全可以先配置好一个 myLog.py，以后需要使用时直接导入到程序中即可。

【示例 4-5】编写 myLog.py。打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi myLog.py
```

myLog.py 的代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import logging
import getpass
import sys


# 定义 MyLog 类
class MyLog(object):
    '''这个类用于创建一个自用的 log '''
    def __init__(self): #类 MyLog 的构造函数
        user = getpass.getuser()
        self.logger = logging.getLogger(user)
        self.logger.setLevel(logging.DEBUG)
        logFile = './' + sys.argv[0][0:-3] + '.log' #日志文件名
        formatter = logging.Formatter('%(asctime)-12s %(levelname)-8s %(name)-10s %(message)-12s')

        '''日志显示到屏幕上并输出到日志文件内'''
        logHand = logging.FileHandler(logFile)
        logHand.setFormatter(formatter)
        logHand.setLevel(logging.ERROR) #只有错误才会被记录到 logfile 中

        logHandSt = logging.StreamHandler()
```

```
        logHandSt.setFormatter(formatter)

        self.logger.addHandler(logHand)
        self.logger.addHandler(logHandSt)

    ''' 日志的 5 个级别对应以下的 5 个函数 '''
    def debug(self,msg):
        self.logger.debug(msg)

    def info(self,msg):
        self.logger.info(msg)

    def warn(self,msg):
        self.logger.warn(msg)

    def error(self,msg):
        self.logger.error(msg)

    def critical(self,msg):
        self.logger.critical(msg)

if __name__ == '__main__':
    mylog = MyLog()
    mylog.debug("I'm debug")
    mylog.info("I'm info")
    mylog.warn("I'm warn")
    mylog.error("I'm error")
    mylog.critical("I'm critical")
```

输入:wq，保存 myLog.py。myLog.py 可以当成一个脚本执行，也可以当成一个模块导入到其他的脚本中执行。在这里是作为脚本使用的，它的作用是将所有的 log 信息显示到屏幕上，错误信息存入到 log 文档中去。执行命令：

```
python myLog.py
cat myLog.log
```

得到的结果如图 4-6 所示。

```
king@debian: ~/code/crawler
king@debian:~/code/crawler$ python myLog.py
2016-06-30 14:35:55,662 DEBUG    king       I'm debug
2016-06-30 14:35:55,663 INFO     king       I'm info
2016-06-30 14:35:55,664 WARNING  king       I'm warn
2016-06-30 14:35:55,665 ERROR    king       I'm error
2016-06-30 14:35:55,666 CRITICAL king       I'm critical
king@debian:~/code/crawler$ cat myLog.log
2016-06-30 14:35:55,665 ERROR    king       I'm error
2016-06-30 14:35:55,666 CRITICAL king       I'm critical
king@debian:~/code/crawler$
```

图 4-6　run myLog.py

【示例 4-6】下面再写一个 testMyLog.py，在程序中导入上图中的 myLog.py 作为模块使用。编写 testMyLog.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testMyLog.py
```

testMyLog.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

from myLog import MyLog

if __name__ == '__main__':
    ml = MyLog()
    ml.debug('I am debug message')
    ml.info('I am info message')
    ml.warn('I am warn message')
    ml.error('I am error message')
    ml.critical('I am critical message')
```

输入:wq，保存 testMyLog.py。testmyLog.py 调用了 myLog.py，将它作为模块使用。执行命令：

```
python testMyLog.py
cat testMyLog.log
```

得到的结果如图 4-7 所示。

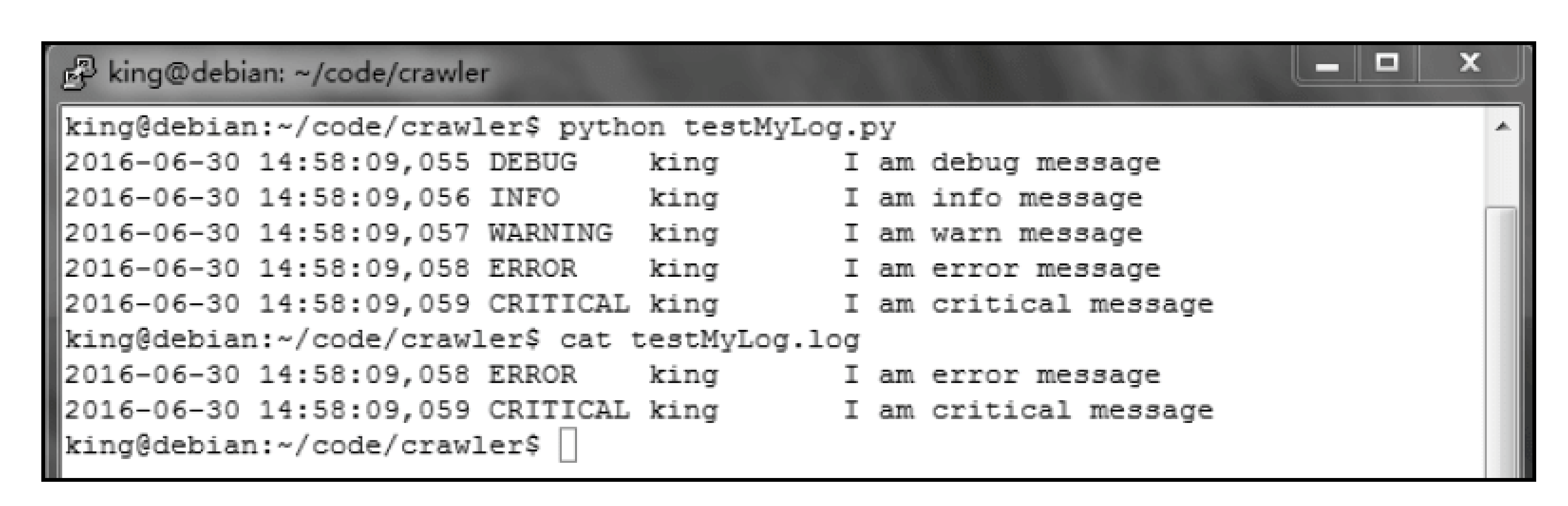


图 4-7　导入自定义模块

在编程时，有时为了查看程序的进度和参数的变化，在程序中间插入了大量的 print。检查完毕后又要逐个删除，费时费力。使用 log 后就简单多了，直接保存为日志文件即可。

4.3 其他有用模块

在编写网络爬虫时，还有一些模块是必不可少的。这些模块使用频率不高，如果不想深究，稍作了解即可。

4.3.1 re 模块（正则表达式操作）

re 模块是文件处理中必不可少的模块。它主要应用于字符串的查找、定位等等。在使用网络爬虫时，即使没有爬虫框架，re 模块配合 urllib2 模块也可以完成简单的爬虫功能。先来看看所谓的正则表达式，以下是 Python 支持的正则表达式元字符和语法。

1．字符

- .: 匹配任意除换行符\n 外的字符，abc 匹配 abc。
- \: 转义字符，使后一个字符改变原来的意思，a\.bc 匹配 a.bc。
- [...]: 字符集（字符类）。对应字符集中的任意字符，第一个字符是^则取反。a[bc]d 匹配 abd,acd。

2．预定义字符集

- \d: 数字[0-9]。
- \D: 非数字[^\d]。
- \s: 空白字符[空格\t\r\n\f\v]。
- \S: 非空白字符[^\s]。
- \w: 单词字符[a-zA-Z0-9_]。
- \W: 非单词字符[^\w]。

3．数量词

- *: 匹配前一个字符 0 或无限次。a1*b 匹配 ab、a1b、a11b……
- +: 匹配前一个字符 1 或无限次。a1*b 匹配 a1b、a11b……
- ?: 匹配前一个字符 0 或 1 次。a1*b 匹配 ab、a1b。
- {m}: 匹配前一个字符 m 次。a1{3}b 匹配 a111b。
- {m,n}: 匹配前一个字符 m 至 n 次。a1{2,3}b 匹配 a11b、a111b。

4．边界匹配

- ^: 匹配字符串开头，如^abc 匹配以 abc 开头的字符串。
- $: 匹配字符串结尾，如 xyz$匹配以 xyz 结尾的字符串。
- \A: 仅匹配字符串开头，如\Aabc。
- \Z: 仅匹配字符串结尾，如 Xyz\Z。

Python 的 re 模块提供了两种不同的原始操作：match 和 search。match 是从字符串的起点开始做匹配，而 search（perl 默认）是对字符串做任意匹配。最常用的几个 re 模块方法如下：

- re.compile(pattern, flags=0)：将字符串形式的正则表达式编译为 Pattern 对象。
- re.search(string[, pos[, endpos]])：从 string 的任意位置开始匹配。
- re.match(string[, pos[, endpos]])：从 string 的开头开始匹配。
- re.findall(string[, pos[, endpos]])：从 string 任意位置开始匹配，返回一个列表。
- re.finditer(string[, pos[, endpos]])：从 string 任意位置开始匹配，返回一个迭代器。一般匹配 findall 就可以了，大数量的匹配还是使用 finditer 比较好。

简单地测试一下，打开 IDLE 执行命令：

```
import re
s = 'I am python modules test for re modules'
re.search('am',s)
re.search('am',s).group()
re.match('am',s)
re.match('I am',s)
re.match('I am',s).group()
re.findall('modules',s)
re.finditer('modules',s)
for sre in re.finditer('modules',s):
 print(sre.group())
```

执行结果如图 4-8 所示。

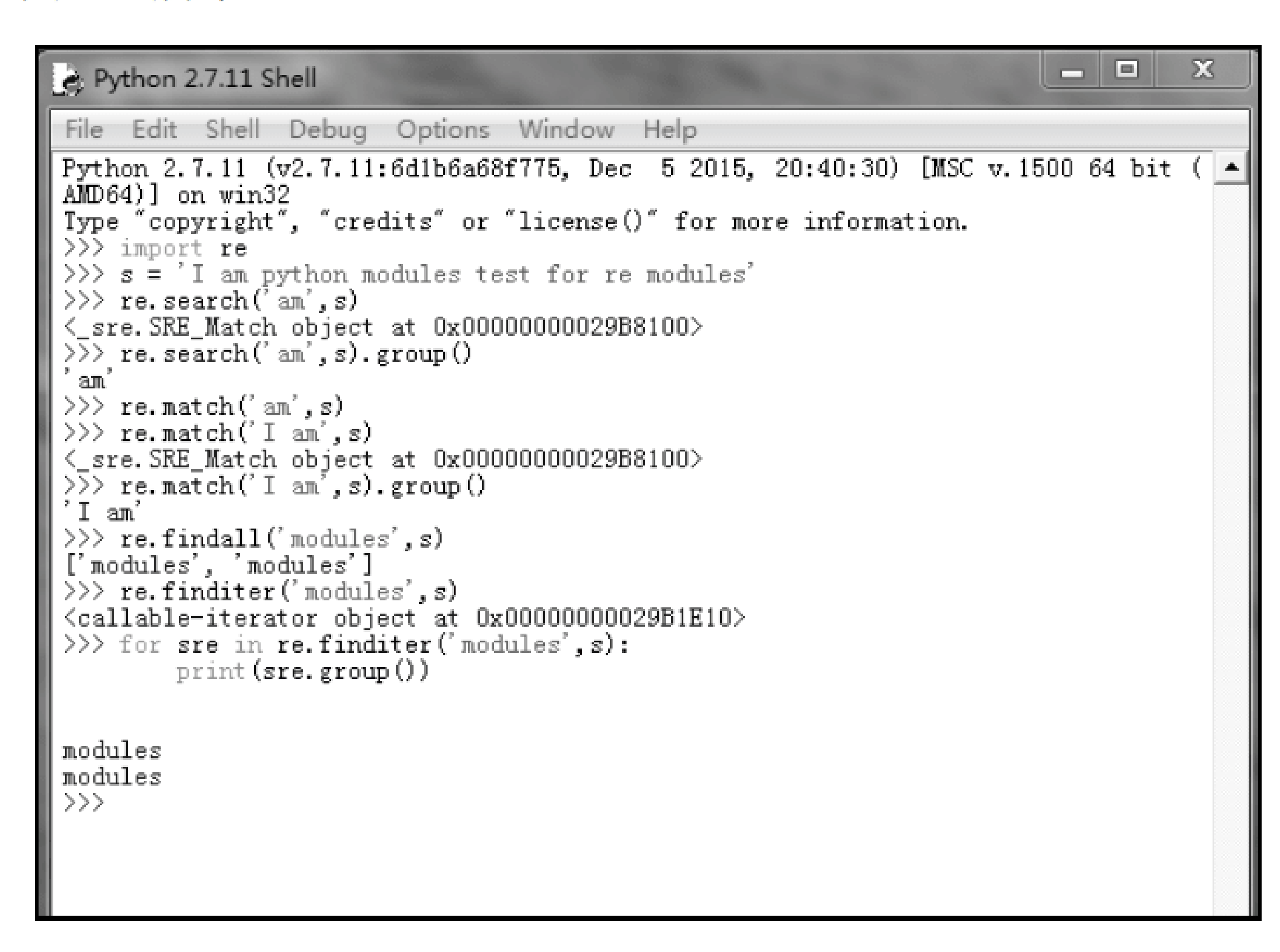


图 4-8　re 匹配字符串

现在用 re 模块和 urllib2 模块来做一个简单的网络爬虫，例如笔者想看看最近的电影院播

放的今日电影。先找找最近的影院，就以金逸影城为例。找到影院页面 http://www.jycinema.com/browsing/Cinemas/Details/1029，先使用 urllib2 模块抓取整个网页，再使用 re 模块获取影视信息。

【示例 4-7】编写 simpleCrawlerNowMoive.py，Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi simpleCrawlerNowMoive.py
```

simpleCrawler.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import re
import urllib2

class TodayMoive(object):
    '''获取金逸影院当日影视 '''
    def __init__(self):
        self.url = 'http://www.jycinema.com/browsing/Cinemas/Details/1029'
        self.timeout = 5
        self.fileName = './todayMoive.txt'
        '''内部变量定义完毕 '''
        self.getMoiveInfo()

    def getMoiveInfo(self):
        response = urllib2.urlopen(self.url,timeout=self.timeout)
        moiveList = re.findall('film-title.*',response.read())
        with open(self.fileName,'w') as fp:
            for moive in moiveList:
                moive = self.subStr(moive)
                print(moive.decode('utf8'))
                fp.write(moive + '\n')

    def subStr(self,st):
        st = st.replace('film-title">','')
        st = st.replace('</h3>','')
        return st


if __name__ == '__main__':
    tm = TodayMoive()
```

输入:wq，保存 simpleCrawlerNowMoive.py。simpleCrawlernowMoive.py 使用 urllib2 模块获取 URL 的返回信息，然后使用 re 模块从结果中过滤得到当日电影的列表，最后显示到屏幕上。执行命令：

```
python simpleCrawlerNowMoive.py
```

得到的结果如图 4-9 所示。

```
king@debian: ~/code/crawler
king@debian:~/code/crawler$ python simpleCrawlerTodayMoive.py
陆垚知马俐 （数字）
忍者神龟2：破影而出 （IMAX 3D）
忍者神龟2：破影而出 （数字3D）
党的女儿尹灵芝 （数字）
古田会议 （数字）
魔轮 （数字）
赏金猎人 （数字）
终极胜利 （数字）
所以……和黑粉结婚了 （数字）
发条城市 （数字）
独立日：卷土重来 （IMAX 3D）
独立日：卷土重来 （数字3D）
惊天魔盗团2 （数字3D）
三人行 （数字）
海底总动员2：多莉去哪儿 （数字3D）
魔兽 （数字3D）
X战警：天启 （数字3D）
king@debian:~/code/crawler$
```

图 4-9　run simpleCrawlerNowMoive.py

看起来不像是网络爬虫，对吗？严格来说这个就是网络爬虫了。只是爬取的内容很简单、也很少罢了。当爬取的内容比较少的时候，网络爬虫也可以这么写。稍微复杂点的、爬取内容多一点的，按照这个方法写那就很痛苦了，这个时候就要用到爬虫框架了。

4.3.2　sys 模块（系统参数获取）

sys 模块，顾名思义就是跟系统相关的模块，这个模块的函数方法不多。最常用的就只有两个。sys.argv 和 sys.exit。sys.argv 返回一个列表，包含了所有的命令行参数；sys.exit 则是退出程序。再就是可以返回当前系统平台。这个模块比较简单，稍作了解即可。

【示例 4-8】编写 testSys.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testSys.py
```

testSys.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

import sys

class ShowSysModule(object):
    '''这个类用于展示 python 标准库中的 sys 模块 '''
    def __init__(self):
        print(u'sys 模块最常用的功能就是获取程序的参数')
        self.getArg()
        print(u'其次就是获取当前的系统平台')
        self.getOs()

    def getArg(self):
```

```
        print(u'开始获取参数的个数')
        print(u'当前参数有 %d 个' %len(sys.argv))
        print(u'这些参数分别是 %s' %sys.argv)

    def getOs(self):
        print(u'sys.platform 返回值对应的平台：')
        print('System\t\t\tPlatform')
        print('Linux\t\t\tlinux2')
        print('Windows\t\t\twin32')
        print('Cygwin\t\t\tcygwin')
        print('Mac OS X\t\tdarwin')
        print('OS/2\t\t\tos2')
        print('OS/2 EMX\t\tos2emx')
        print('RiscOS\t\t\triscos')
        print('AtheOS\t\t\tatheos')
        print('\n')
        print(u'当前的系统为 %s' %sys.platform)

if __name__ == '__main__':
    ssm = ShowSysModule()
```

输入:wq，保存 testSys.py。testSys.py 顾名思义用于测试 sys 模块。该脚本将获取系统平台、Python 脚本参数的个数、参数的值等信息。执行命令：

```
python testSys.py 1 2 3 4 5
```

得到的结果如图 4-10 所示。

```
king@debian:~/code/crawler$ python testSys.py 1 2 3 4 5
sys模块最常用的功能就是获取程序的参数
开始获取参数的个数
当前参数有 6 个
这些参数分别是 ['testSys.py', '1', '2', '3', '4', '5']
其次就是获取当前的系统平台
sys.platform返回值对应的平台:
System                  Platform
Linux                   linux2
Windows                 win32
Cygwin                  cygwin
Mac OS X                darwin
OS/2                    os2
OS/2 EMX                os2emx
RiscOS                  riscos
AtheOS                  atheos


当前的系统为 linux2
king@debian:~/code/crawler$
```

图 4-10　run testSys.py

sys 模块用处不多，但也需要熟悉。顾名思义，它的主要作用就是返回系统信息。

4.3.3　time 模块（获取时间信息）

Python 中的 time 模块是跟时间相关的模块。这个模块用得最多的地方可能就是计时器了。本节只介绍最常用的几个函数。

- time.time()：返回当前的时间戳。
- time.localtime([secs])：默认将当前时间戳转换成当前时区的 struct_time。
- time.sleep(secs)：计时器。
- Time.strftime(format[, t])：把一个 struct_time 转换成格式化的时间字符串。这个函数支持的格式符号如表 4-1 所示。

表 4-1　时间字符串支持的格式符号

格式	含义
%a	本地（locale）简化星期名称
%A	本地完整星期名称
%b	本地简化月份名称
%B	本地完整月份名称
%c	本地相应的日期和时间表示
%d	一个月中的第几天（01～31）
%H	一天中的第几个小时（24 小时制，00～23）
%I	第几个小时（12 小时制，01～12）
%j	一年中的第几天（001～366）
%m	月份（01～12）
%M	分钟数（00～59）
%p	本地 am 或者 pm 的响应符
%S	秒（01～61）
%U	一年中的星期数
%w	一个星期中的第几天（0～6，0 是星期天）
%W	和%U 基本相同，不同的是%W 以星期一为一个星期的开始
%x	本地相应日期
%X	本地相应时间
%y	简化的年份（00－99）
%Y	完整的年份
%Z	时区的名字（如果不存在为空字符）
%%	‘%'字符
备注： 在使用 strptime()函数时，%p 和%I 配合使用才有效。 %S 中的秒是 0~61，闰年中的秒占两秒。 在使用 strtime()函数时，只有当年中的周数和天数被确定时，%U 和%W 才被计算。	

简单地测试一下，在 Windows 中打开 IDLE，执行命令：

```
import time
time.time()
time.localtime()
for i in xrange(5):
 time.sleep(1)
 print i
time.strftime('%Y-%m-%d %X' ,time.localtime())
```

执行结果如图 4-11 所示。

```
Python 2.7.11 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500 64 bit (
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.time()
1467283311.796
>>> time.localtime()
time.struct_time(tm_year=2016, tm_mon=6, tm_mday=30, tm_hour=18, tm_min=41, tm_s
ec=57, tm_wday=3, tm_yday=182, tm_isdst=0)
>>> for i in xrange(5):
	time.sleep(1)
	print i


0
1
2
3
4
>>> time.strftime('%Y-%m-%d %X',time.localtime())
'2016-06-30 18:42:44'
>>>
```

图 4-11　show time module

【示例 4-9】做个简单的程序，实验一下 time 模块。编写 testTime.py，打开 Putty 连接到 Linux，执行命令：

```
cd code/crawler
vi testTime.py
```

testTime.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


import time
from myLog import MyLog
''' 这里的 myLog 是自建的模块，处于该文件的同一目录下'''

class TestTime(object):
    def __init__(self):
```

```
        self.log = MyLog()
        self.testTime()
        self.testLocaltime()
        self.testSleep()
        self.testStrftime()

    def testTime(self):
        self.log.info(u'开始测试 time.time()函数')
        print(u'当前时间戳为：time.time() = %f' %time.time())
        print(u'这里返回的是一个浮点型的数值，它是从 1970 纪元后经过的浮点秒数')
        print('\n')

    def testLocaltime(self):
        self.log.info(u'开始测试 time.localtime()函数')
        print(u'当前本地时间为：time.localtime() = %s' %time.localtime())
        print(u'这里返回的是一个 struct_time 结构的元组')
        print('\n')

    def testSleep(self):
        self.log.info(u'开始测试 time.sleep()函数')
        print(u'这是个计时器：time.sleep(5)')
        print(u'闭上眼睛数上 5 秒就可以了')
        time.sleep(5)
        print('\n')

    def testStrftime(self):
        self.log.info(u'开始测试 time.strftime()函数')
        print(u'这个函数返回的是一个格式化的时间')
        print('time.strftime("%%Y-%%m-%%d %%X",time.localtime()) = %s' %time.strftime("%Y-%m-%d %X",time.localtime()))
        print('\n')


if __name__ == '__main__':
    tt = TestTime()
```

输入:wq，保存 testTime.py。testTime.py 测试了 time 模块的定时器功能，并显示当前时间。执行命令：

```
python testTime.py
```

得到的结果如图 4-12 所示。

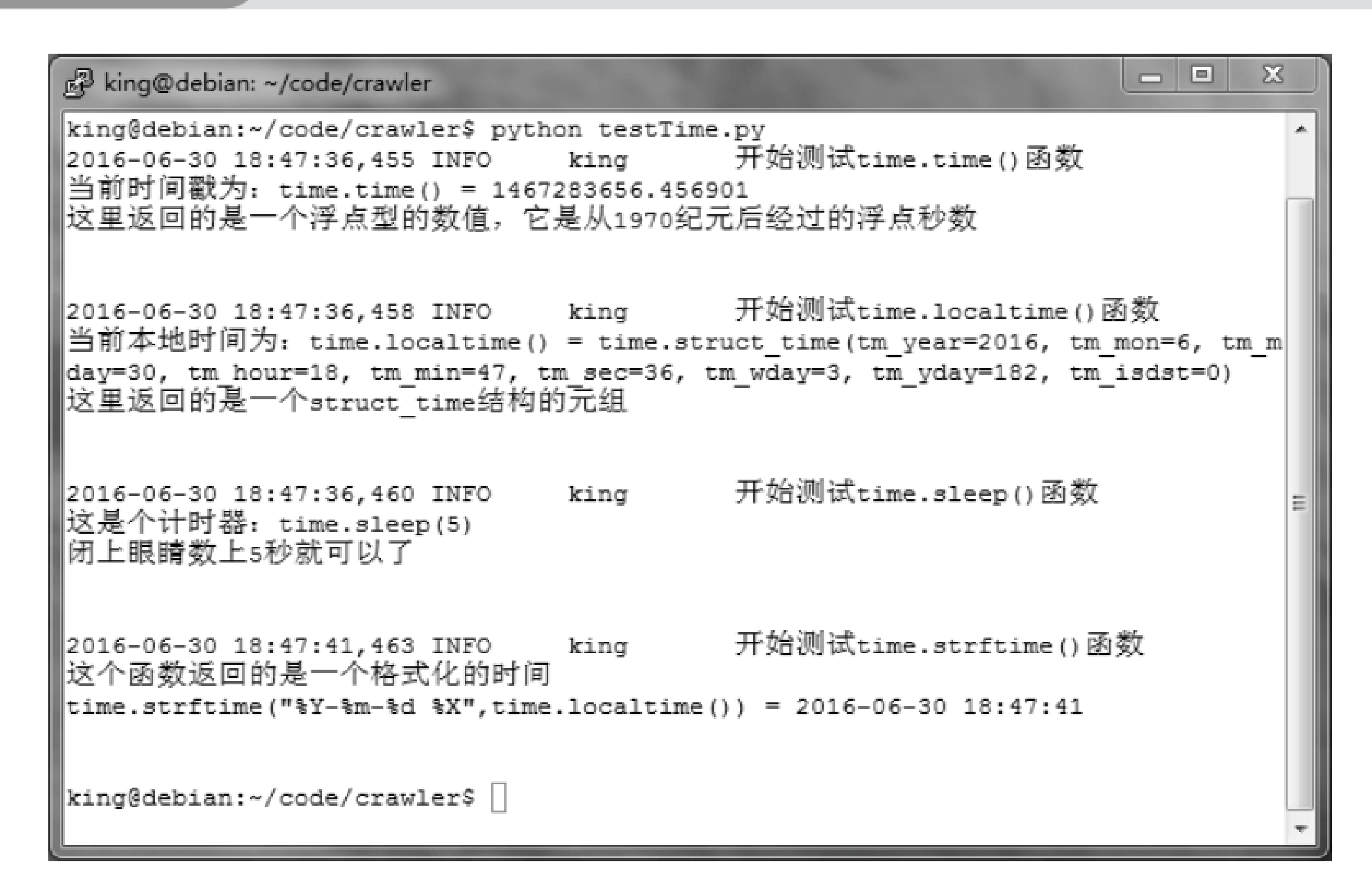


图 4-12 run testTime.py

time 模块还有很多函数，最常用的还是计时器，其次就是做时间戳了。

4.4 本章小结

如果只想大致了解 Python 网络爬虫，熟悉了这些模块也就差不多了。即使爬虫中可能还需要其他模块，也无须担心，大多也只是需要模块中的某个函数。完全可以把它当成函数使用，这样就简单多了。Python 2 的标准模块差不多有 300 个。熟悉所有的模块，既没必要，也不可能。用到哪个模块就熟悉哪个模块，就可以了。

第 5 章

◀ Scrapy爬虫框架 ▶

网络爬虫的最终目的就是从网页中截取自己所需要的内容。最直接的方法当然是用 urllib2 请求网页得到结果，然后使用 re 取得所需的内容。但如果所有的爬虫都这样写，工作量未免太大了点，所以才有了爬虫框架。

Python 下的爬虫框架不少，笔者认为最简单的就要数 Scrapy 了。首先它的资料比较全，网上的指南、教程都比较多。其次它够简单，只要按需填空即可，简简单单地就能获取到所需的内容，非常方便。

5.1 安装 Scrapy

Scrapy 的官网是 http://scrapy.org/，目前最新版本的 Scrapy 是 Scrapy 1.1。Scrapy 的安装方式很多。官网上就给出了 4 种安装方法，PyPI、Conda、APT 、Source 安装。这里笔者只演示最常用的两种安装方法。

5.1.1 Windows 下安装 Scrapy 环境

Windows 下安装 Scrapy 除了不能使用 APT 安装外，其他的三种方法都是可以的。这里笔者选择了最简单的 PyPI 安装，也就是 pip 安装。pip 安装 Scrapy 的前提条件是已经安装好了 Python，并配置好了 pip 源。如果这些条件已经具备，安装 Scrapy 只需要打开 cmd，执行一条命令而已。打开 cmd 并执行命令：

```
pip install scrapy
```

执行结果如图 5-1 所示。

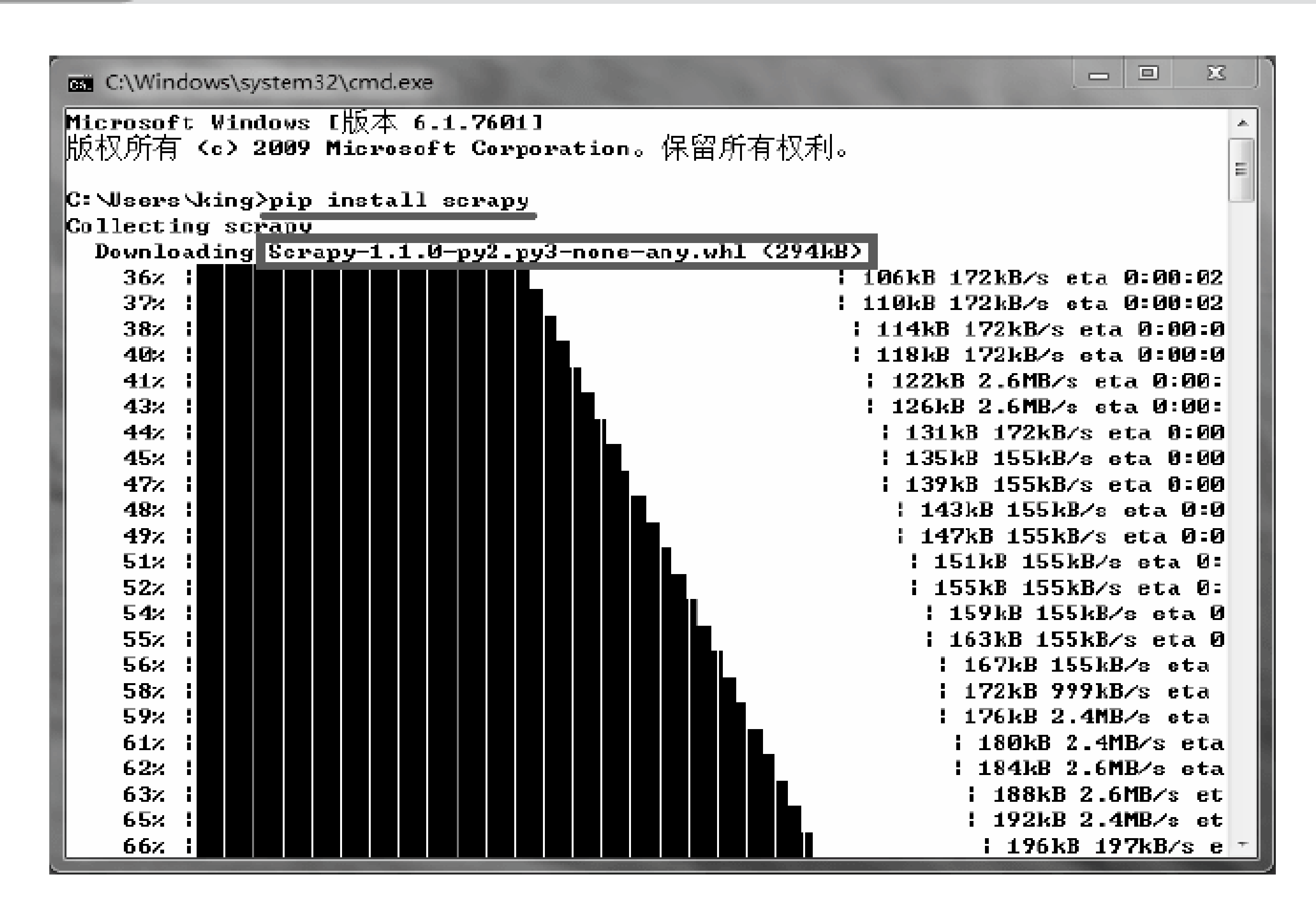


图 5-1　使用 pip 安装 scrapy

Windows 下的 Scrapy 和 Linux 下的 Scrapy 是完全一样的。但笔者一般都会在 Linux 下使用 Scrapy。Linux 的优点是有很多辅助工具可供使用，而 Windows 的优点就只剩下有很多 IDE 可供选择了。而在这里使用 Scrapy 框架时，IDE 却是最无关紧要的参考选项了。

5.1.2　Linux 下安装 Scrapy

Linux 下安装 Scrapy 官网上推荐的 4 种方法都可以。从上一小节看，Windows 下使用了 pip 安装 scrapy 已经很简单了，Linux 下使用更简单的方法 APT 来安装。Scrapy 官方推荐的 APT 安装还需要在 Linux 中添加 apt 源。得益于 Debian 庞大的软件库，这一步完全可以省略。直接使用 Debian Linux 的 apt-get 安装 Scrapy，简单方便而且将 Scrapy 列入了 apt 包管理器中。以后升级、删除都很方便。安装 Scrapy 也是只需要一条命令而已，执行命令：

```
apt-get install python-scrapy
```

执行结果如图 5-2 所示。



图 5-2　使用 apt-get 安装 scrapy

因为 Debian 安装软件选择的都是最成熟稳定的版本，而不是最新的版本。所以这里安装的 Scrapy 并不是最新的 1.1 版，如图 5-3 所示。

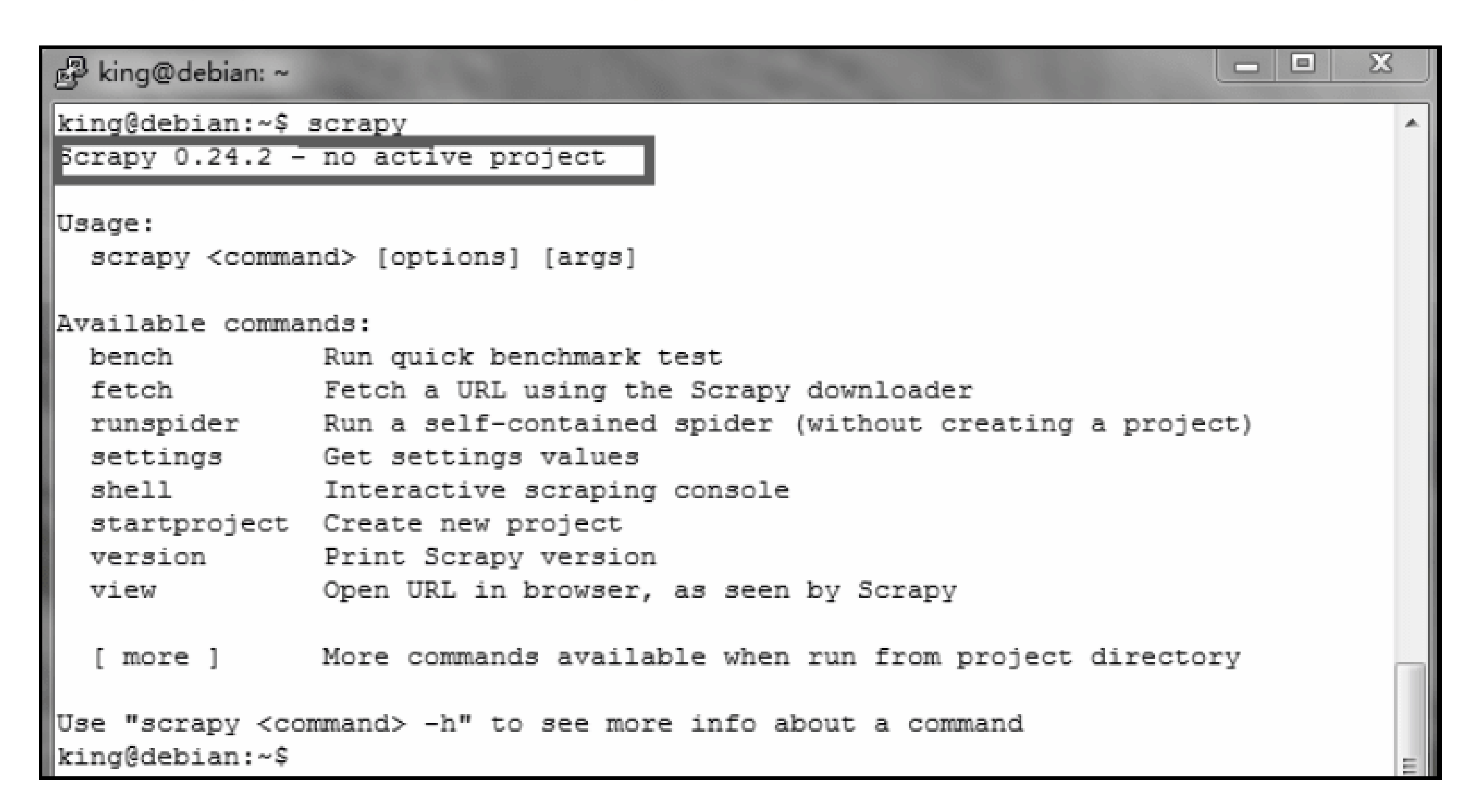


图 5-3　Scrapy 版本

这两个版本并没有什么本质的不同，我们可以放心使用。

5.1.3　vim 编辑器

本章的 Scrapy 项目主要是在 Linux 下运行。在 Linux 下最强大的 IDE 是 Eclipse，但最方便的还是 vim（vim 是 vi 的强化版，而 vi 是所有 Linux 发行版本都默认安装的）。

vim 是一个文本编辑器，在上手时可能稍有点麻烦。它有一些快捷键和命令是必须要记住的，可以边使用边记忆。等熟悉了 vim 的操作方法，就会发现文本编辑是如此简单。对不同的编程语言配合不同的插件，可以将 vim 配置成为一个专属的 IDE。

vim 安装非常简单，使用 Putty 登录 Linux 后，以 root 用户执行命令：

```
apt-get install vim
```

vim 的配置文件是/etc/vim/vimrc 和/home/`user`/.vim/vimrc（对于用户 king 来说就是/home/king/.vim/vimrc）。前者是系统配置文件，后者是用户的配置文件。两者相冲突，则以后者为主。

vim 的配置项很多这里不一一列举，为了编写 Python 程序方便，这里只修改最简单的设置。笔者的 vimrc 文件如下：

```
1 set tabstop=4
2 set number
3 set noexpandtab
```

第 1 行的设置是将 tabstop 设置成 4 个空格，第 2 行是显示行号，第 3 行不将 tabstop 转换成空格。

如果经常在 Linux 写 Python 程序，可以到 github 上下载 vim 变身 Python IDE 的配置文件。仔细调试一下，vim IDE 不比 Windows 下的 Python IDE 差。

5.2 Scrapy 选择器 XPath 和 CSS

在使用 Scrapy 爬取数据前需要先了解 Scrapy 的选择器。在前面章节曾经提过，网络爬虫原理就是获取网页返回，然后提取所需的内容。获取网页返回很简单，重点就在提取内容上。如何提取？使用 Python 的 re 模块，前面的章节中已经尝试过了。简单网页用 re 模块提取可以将就，复杂一点的提取内容就麻烦了。不是说完全不可以，但是有简单的方法又何必去自己编写新方法呢。

Scrapy 提取数据有自己的一套机制。它们被称作选择器（seletors），通过特定的 XPath 或者 CSS 表达式来“选择”HTML 文件中的某个部分。

XPath 是一门用来在 XML 文件中选择节点的语言，也可以用在 HTML 上。CSS 是一门将 HTML 文档样式化的语言。选择器由它定义，并与特定的 HTML 元素的样式相关联。

Scrapy 的选择器构建于 lxml 库之上，这意味着它们在速度和解析准确性上非常相似。所以看你喜欢那种选择器就使用那种吧，它们从效率上看是完全没有区别。

5.2.1 XPath 选择器

XPath 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。XPath 含有超过 100 个内建的函数。这些函数用于字符串值、数值、日期和时间比较、节点和 QName 处理、序列处理、逻辑值等等。在网络爬虫中只需要利用 XPath“采集”数据，如果想深入研究可参考 www.w3school.com.cn 中的 XPath 教程。

在 XPath 中，有 7 种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及文档节点（或称为根节点）。XML 文档是被作为节点树来对待的。树的根被称为文档节点或者根节点。

【示例 5-1】做个简单的 XML 文件，以便演示。执行命令：

```
cd
cd code/crawler
mkdir -pv scrapyProject/seletors
cd scrapyProject/seletors
vi superHero.xml
```

在这里创建了 scrapy 的工作目录 scrapyProject，并在该目录下创建了选择器的工作目录 seletors。在该目录下创建选择器的演示文件 superHero.xml。superHero.xml 的代码如下：

```
<superhero>
<class>
    <name lang="en">Tony Stark </name>
    <alias>Iron Man </alias>
    <sex>male </sex>
    <birthday>1969 </birthday>
```

```
    <age>47 </age>
</class>
<class>
    <name lang="en">Peter Benjamin Parker </name>
    <alias>Spider Man </alias>
    <sex>male </sex>
    <birthday>unknow </birthday>
    <age>unknown </age>
</class>
<class>
    <name lang="en">Steven Rogers </name>
    <alias>Captain America </alias>
    <sex>male </sex>
    <birthday>19200704 </birthday>
    <age>96 </age>
</class>
</superhero>
```

很简单的一个 XML 文件，在浏览器中打开这个文件，如图 5-4 所示。

```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

▼<superhero>
  ▼<class>
      <name lang="en">Tony Stark</name>
      <alias>Iron Man</alias>
      <sex>male</sex>
      <birthday>1969</birthday>
      <age>47</age>
    </class>
  ▼<class>
      <name lang="en">Peter Benjamin Parker</name>
      <alias>Spider Man</alias>
      <sex>male</sex>
      <birthday>unknow</birthday>
      <age>unknown</age>
    </class>
  ▼<class>
      <name lang="en">Steven Rogers</name>
      <alias>Captain America</alias>
      <sex>male</sex>
      <birthday>19200704</birthday>
      <age>96</age>
    </class>
  </superhero>
```

图 5-4　选择器演示文件 superHero.xml

后面的选择器都以该文件为示例。在 superHero.xml 中，<superhero>是文档节点，<alias>Iron Man</alias>是元素节点，lang="en"是属性节点。

从节点的关系来看。第一个 Class 节点是 name、alias、sex、birthday、age 节点的父节点（Parent）。反过来说，name、alias、sex、birthday、age 节点是第一个 Class 节点的子节点（Childer）。name、alias、sex、birthday、age 节点之间互为同胞节点（sibling）。这只是个最简单的例子，如果节点的“深度”足够，还会有先辈节点（Ancestor）和后代节点（Descendant）。

XPath 使用路径表达式在 XML 文档中选取节点。表 5-1 中列出了最常用的路径表达式。

表 5-1　路径表达式

表达式	描述
nodeName	选取此节点的所有子节点
/	从根节点选取
//	从匹配选择的当前节点选择文档中的节点，不考虑它们的位置
.	选取当前节点
..	选取当前节点的父节点
@	选取属性
*	匹配任何元素节点
@*	匹配任何属性节点
Node()	匹配任何类型的节点

下面用 XPath 选择器来“采集”XML 文件中所需的内容，先做好准备工作。执行命令：

```
python
from scrapt.selector import Selector
with open(‘./superHero.xml’,’r’) as fp:
 body = fp.read()
Selector(text=body).xpath(‘/*’).extract()
```

首先启动 Python，导入 scrapy.selector 模块中的 Selector，打开 superHero.xml 文件，并将其内容写入到 body 变量中，最后使用 XPath 选择器显示 superHero.xml 文件中的所有内容。执行结果如图 5-5 所示。

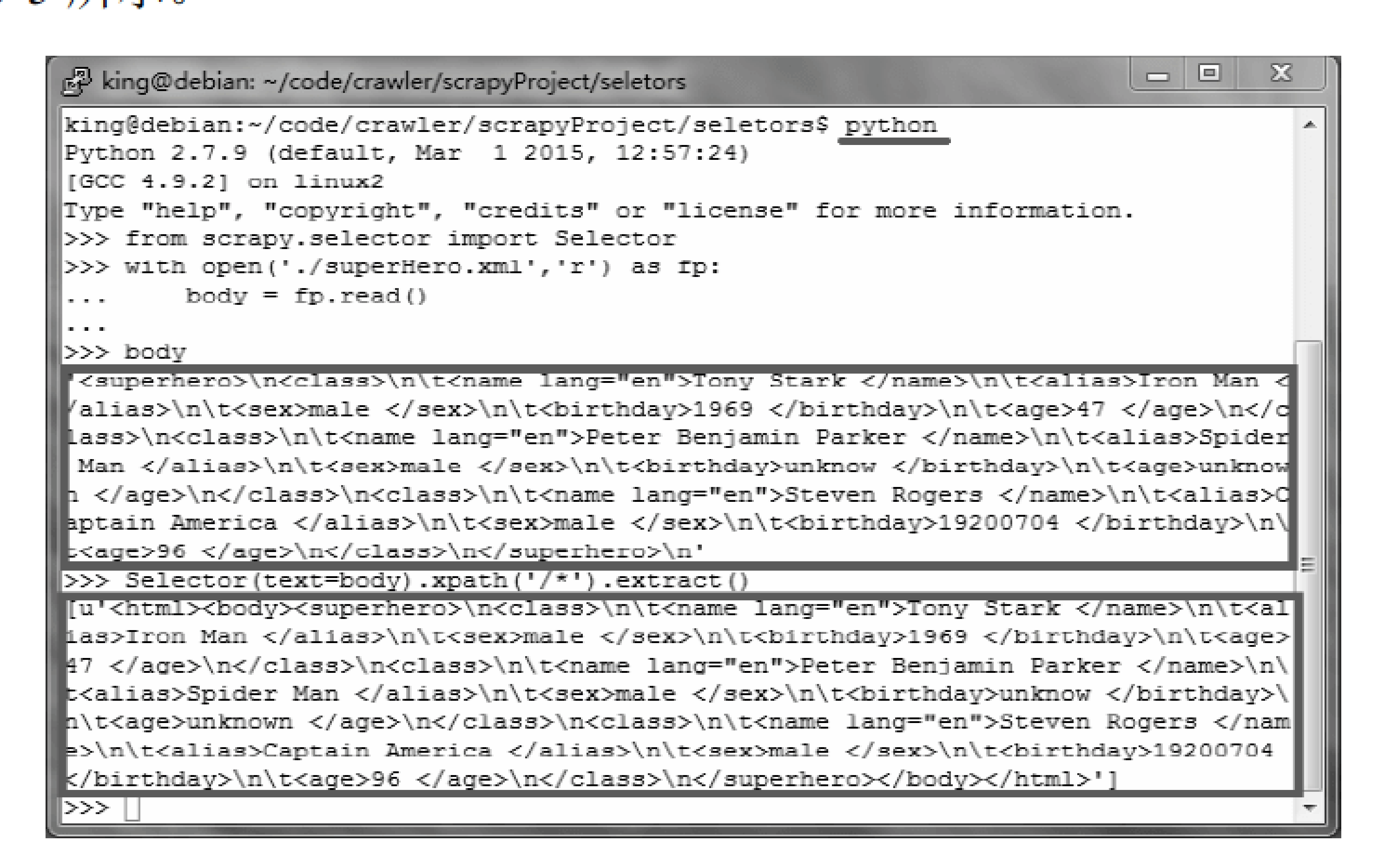


图 5-5　XPath 选择器准备工作



选择器在从根节点选择所有节点时得到的数据和直接从文件中读取的数据有点不一样。因为示例文件并不是一个标准的 html 文件，所以在选择器中被自动添加了<html>和<body>标签。也就是说在选择器看来，示例文件的根节点并不是<superhero>，而是<html>。

好了，现在来看如何使用 XPath 选择器“收集”数据，如图 5-6 所示。



图 5-6　XPath 选择器收集数据

XPath 中最常用的几个方法就是如此了，非常简单。“隐藏”得不太深的数据直接用 XPath 选择器挑选数据就可以了。复杂一点的，用配套选择就能很方便地搞定。只要有点耐心，再复杂的数据也可以分离出来。

5.2.2　CSS 选择器

CSS，看起来很眼熟是不是？没错，就是你已经知道的那个 CSS——层叠样式表。CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明。

```
selector {declaration1; declaration2; ... declarationN }
```

CSS 是网页代码中非常重要的一环，即使不是专业的 Web 从业人员，也有必要认真学习一下。这里只简略介绍一下与爬虫密切相关的选择器，表 5-2 中列出了 CSS 经常使用的几个选择器。

表 5-2　CSS 选择器

.class	.intro	选择 class=“intro”的所有元素
#id	#firstname	选择 id=“firstname”的所有元素
*	*	选择所有元素
element	p	选择所有<p>元素
element,element	div,p	选择所有<div>元素和所有<p>元素
element element	div p	选择<div>元素内部的所有 p 元素
[attribute]	[target]	选择带有 target 属性的所有元素
[attribute=value]	[target=_blank]	选择 target=“_blank”的所有元素

与 XPath 选择器相比较，CSS 选择器稍微复杂一点，但其强大的功能弥补了这点缺陷。下面就来试验一下 CSS 选择器如何收集数据，如图 5-7 所示。

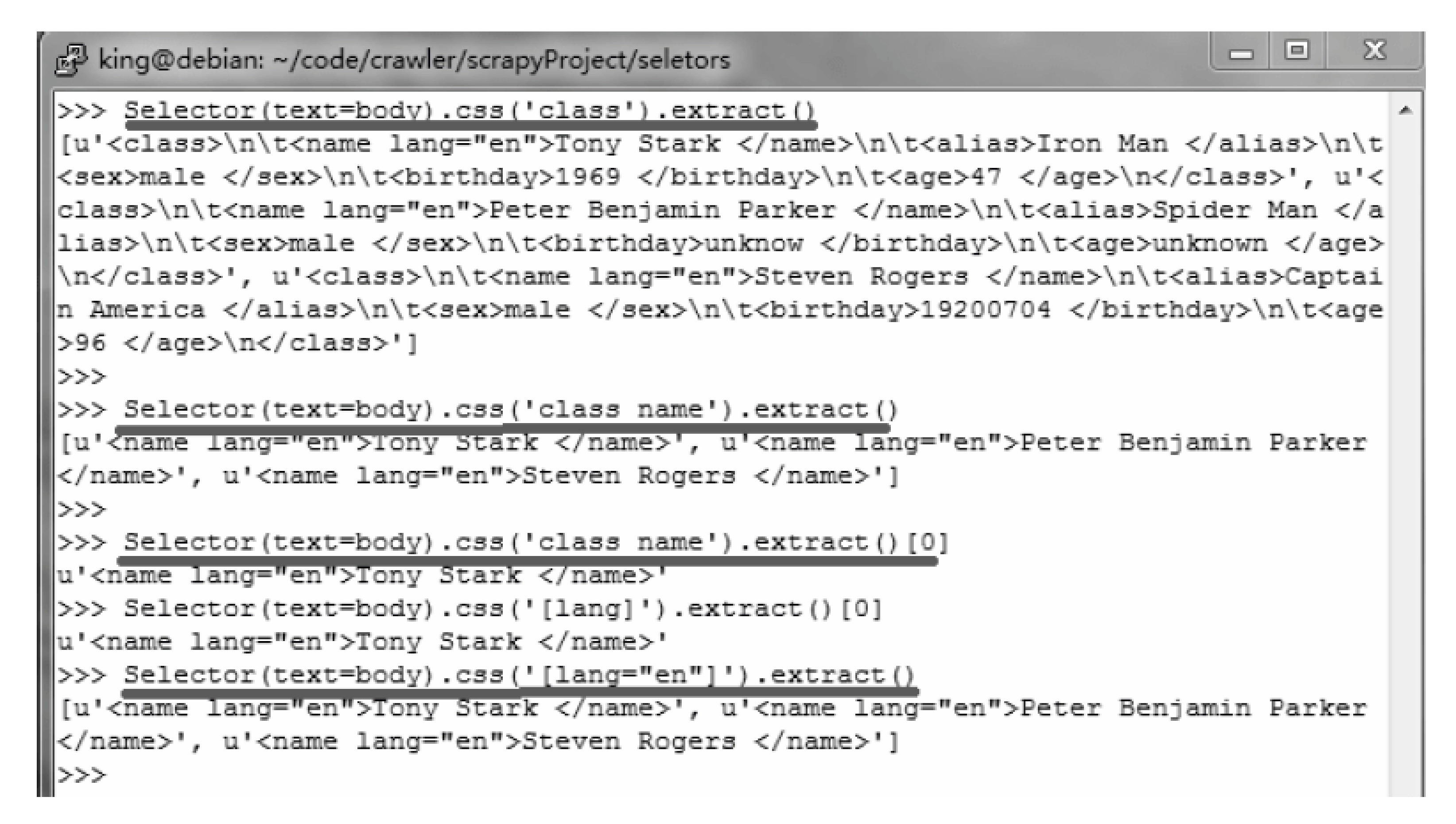


图 5-7　CSS 选择器收集数据

因为 CSS 选择器和 XPath 选择器都可以嵌套使用，所以它们可以互相嵌套，这样一来收集数据会更加方便。

5.2.3　其他选择器

XPath 选择器还有一个.re()方法，用于通过正则表达式来提取数据。然而，不同于使用.xpath()或者.css()方法，.re()方法返回 unicode 字符串的列表，所以无法构造嵌套式的.re()调用。使用方法如图 5-8 所示。

```
>>>
>>> Selector(text=body).xpath('/html/body/superhero/class[1]').re('>.*?<')
[u'>Tony Stark <', u'>Iron Man <', u'>male <', u'>1969 <', u'>47 <']
>>> 
```

图 5-8　re 选择器收集数据

这种方法并不常用。个人觉得还不如在程序中添加代码，直接用 re 模块方便。

因为 Scrapy 选择器建于 lxml 之上，所以它也支持一些 EXSLT 扩展，但这里就不做说明了，有兴趣的读者可以自行 Google。

5.3　Scrapy 爬虫实战一：今日影视

还记得前面章节中用 re 模块操作爬虫在金逸影城的网站中爬取当日影视信息的例子吗？实际上用 Scrapy 来爬取会简单得多。打个比方，前面章节中使用 re 模块爬取当日影视相当于是做作文，而使用 Scrapy 来爬取就相当于做填空题，只需要把相应的要求填入空白框里就可以了。

5.3.1　创建 Scrapy 项目

似乎所有的框架，开始的第一步都是从创建项目开始的，Scrapy 也不例外。在这之前要说明的是 Scrapy 项目的创建、配置、运行……默认都是在终端下操作的。不要觉得很难，其实它真的非常简单，填空题而已。如果实在是无法接受，也可以花点心思配置好 Eclipse，在这个万能 IDE 下操作。个人推荐还是在终端操作比较好，虽然开始可能因为不熟悉而出现很多错误，不过人类不就是在错误中前进吗？错多了，印象深刻了，也就自然学会了。打开 Putty 连接到 Linux，开始创建 Scrapy 项目。执行命令：

```
cd
cd code/crawler/scrapyProject/
scrapy startproject todayMoive
tree todayMoive
```

执行结果如图 5-9 所示。

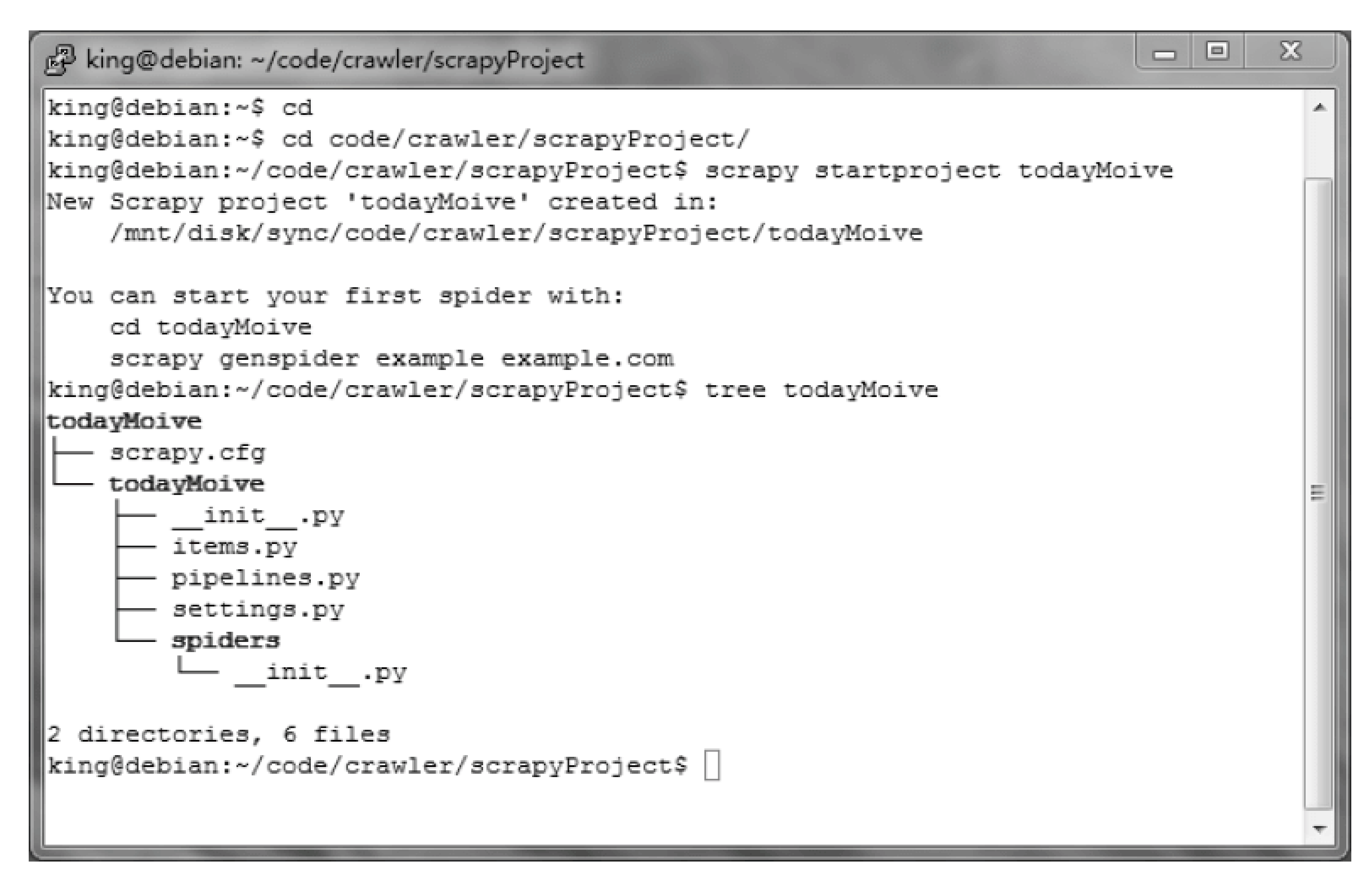


图 5-9　创建 todayMoive 项目



tree 命令将以树形结构显示文件目录结构。tree 命令默认情况下是没有安装的，可以执行命令 apt-get install tree 来安装这个命令。

这里可以很清楚地看到 todayMoive 目录下的所有子文件和子目录。至此 Scrapy 项目 todayMoive 基本上完成了。按照 Scrapy 的提示信息，可以通过 Scrapy 的 Spider 基础模版顺便建立一个基础的爬虫。相当于把填空题打印到试卷上，等待填空了。当然，也可以不用 Scrapy 命令建立基础爬虫，如果非要体验一下 DIY 也是可以的。这里我们还是怎么简单怎么来吧，按照提示信息，在该终端中执行命令：

```
cd todayMoive
scrapy genspider wuHanMoiveSpider jycinema.com
```

执行结果如图 5-10 所示。

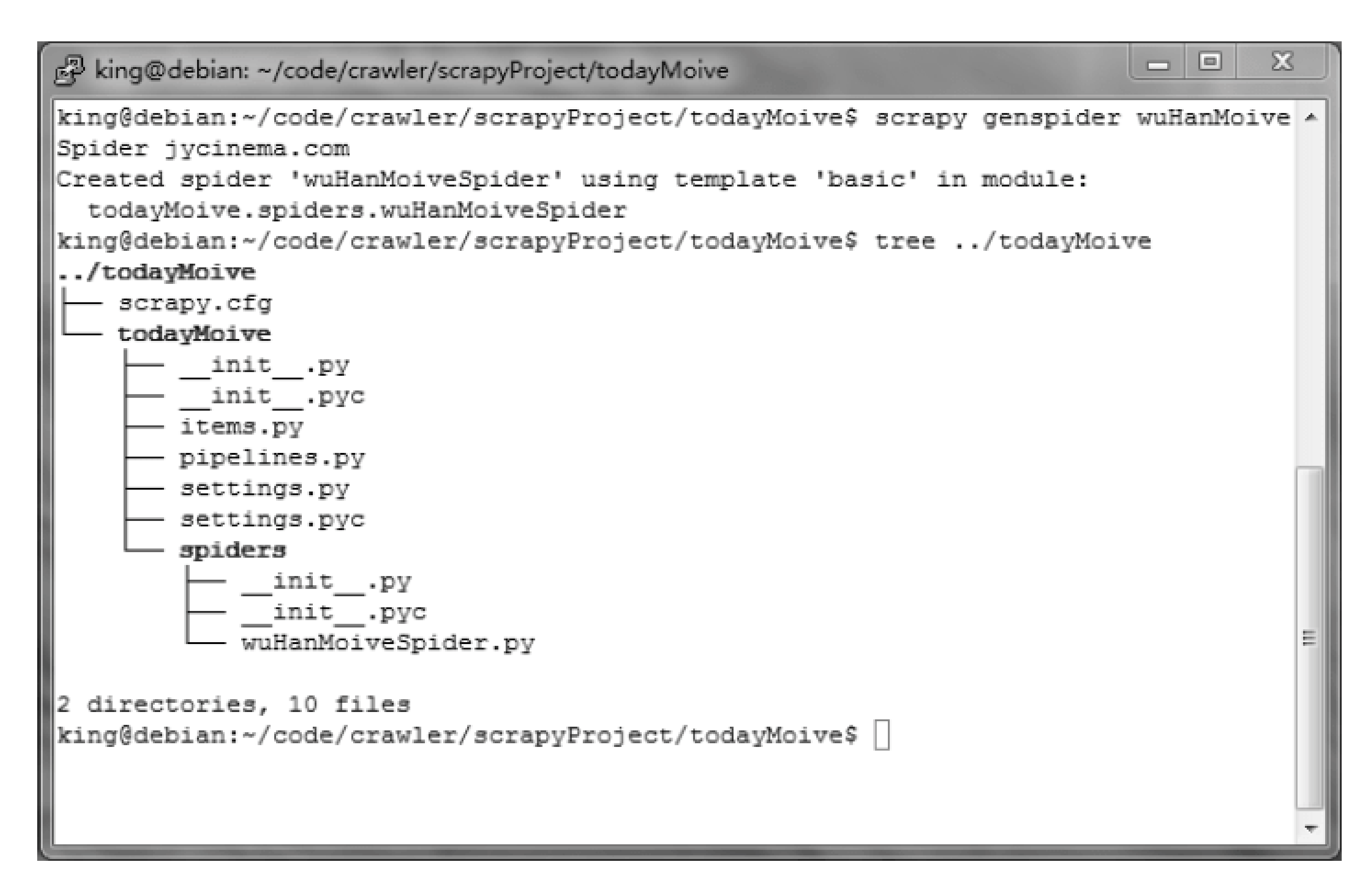


图 5-10　创建基础爬虫

至此，一个最基本的爬虫项目已经建立完毕了，它包含了一个 Scrapy 爬虫所需的基础文件。到这一步可以说填空题已准备完毕，后面的工作就纯粹是填空了。上图中第一行文字 scrapy genspider 是一个命令，也是 scrapy 最常用的几个命令之一，它的使用方法如图 5-11 所示。

```
king@debian: ~/code/crawler/scrapyProject/todayMoive
king@debian:~/code/crawler/scrapyProject/todayMoive$ scrapy genspider -h
Usage
=====
  scrapy genspider [options] <name> <domain>

Generate new spider using pre-defined templates

Options
=======
--help, -h              show this help message and exit
--list, -l              List available templates
--edit, -e              Edit spider after creating it
--dump=TEMPLATE, -d TEMPLATE
                        Dump template to standard output
--template=TEMPLATE, -t TEMPLATE
                        Uses a custom template.
--force                 If the spider already exists, overwrite it with the
                        template

Global Options
--------------
--logfile=FILE          log file. if omitted stderr will be used
--loglevel=LEVEL, -L LEVEL
                        log level (default: DEBUG)
```

图 5-11　scrapy genspider 命令帮助

因此，刚才的命令意思是使用 scrapy genspider 命令创建一个名字为 wuHanMoiveSpider 的爬虫脚本。这个脚本搜索的域为 jycinema.com。

5.3.2　Scrapy 文件介绍

Scrapy 项目的所有文件都已经到位了，如上图 5-10 所示，下面来看看各个文件的作用。首先最顶层的那个 todayMoive 文件夹是项目名，这个没什么好说的。

在第二层中是一个与项目同名的文件夹 todayMoive 和一个文件 scrapy.cfg，这里与项目

同名的文件夹 todayMoive 是模块（也可以叫做包的），所有的项目代码都在这个模块（文件夹或者叫包）内添加。而 scrapy.cfg 文件，顾名思义它是整个 Scrapy 项目的配置文件。来看看这个文件里有些什么。Scrapy.cfg 文件内容如下：

```
# Automatically created by: scrapy startproject
#
# For more information about the [deploy] section see:
# http://doc.scrapy.org/en/latest/topics/scrapyd.html
[settings]
default = todayMoive.settings
[deploy]
#url = http://localhost:6800/
project = todayMoive
```

除去以“#”为开头的注释行，整个文件只声明了两件事。一是定义默认设置文件的位置为 todayMoive 模块下的 settings 文件，二是定义项目名称为 todayMoive。

在第三层中有 6 个文件和一个文件夹（实际上这也是个模块）。看起来很多。实际上有用的也就 3 个文件，分别是 items.py、pipelines.py、settings.py。其他的 3 个文件中，以 pyc 结尾的是同名 Python 程序编译得到的字节码文件，settings.pyc 是 settings.py 的字节码文件，__init__.pyc 是__init__.py 的字节码文件。据说用来加快程序的运行速度，可以忽视。至于__init__.py 文件，它是个空文件，里面什么都没有。在此处唯一的作用就是将它的上级目录变成了一个模块。也就是说第二层的 todayMoive 模块下，如果没有__init__.py 文件。那么 todayMoive 就只是一个单纯的文件夹。在任何一个目录下添加一个空的__init__.py 文件，就会将该文件夹编程模块化，可以供 Python 导入使用。

有用的这 3 个文件中。settings.py 是上层目录中 scrapy.cfg 定义的设置文件。settings.py 的内容如下：

```
# -*- coding: utf-8 -*-
# Scrapy settings for todayMoive project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#
BOT_NAME = 'todayMoive'
SPIDER_MODULES = ['todayMoive.spiders']
```

```
NEWSPIDER_MODULE = 'todayMoive.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'todayMoive (+http://www.yourdomain.com)'
```

items.py 文件的作用是定义爬虫最终需要哪些项，items.py 的内容如下：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class TodaymoiveItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    pass
```

pipelines.py 文件的作用是扫尾。Scrapy 爬虫爬取了网页中的内容后，这些内容怎么处理就取决于 pipelines.py 如何设置了。pipeliens.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html


class TodaymoivePipeline(object):
    def process_item(self, item, spider):
        return item
```

第二层中还有一个 spiders 的文件夹。仔细看一下，在该目录下也有个__init__.py 文件，说明这个文件夹也是一个模块。在该模块下是本项目中所有的爬虫文件。

第三层中有 3 个文件，__init__.py、__init__.pyc、wuHanMoiveSpider.py。前两个文件刚才已经介绍过了，基本不起作用。wuHanMoiveSpider.py 文件是刚才用 scrapy genspider 命令创建的爬虫文件。wuHanMoiveSpider.py 文件内容如下：

```
# -*- coding: utf-8 -*-
```

```
import scrapy


class WuhanmoivespiderSpider(scrapy.Spider):
    name = "wuHanMoiveSpider"
    allowed_domains = ["jycinema.com"]
    start_urls = (
        'http://www.jycinema.com/',
    )

    def parse(self, response):
        pass
```

在本次的爬虫项目示例中，需要修改、填空的只有 4 个文件，它们分别是 items.py、settings.py、pipelines.py、wuHanMoiveSpider.py。其中 items.py 决定爬取哪些项目，wuHanMoiveSpider.py 决定怎么爬，settings.py 决定由谁去处理爬取的内容，pipelines.py 决定爬取后的内容怎样处理。

5.3.3 Scrapy 爬虫编写

My first scrapy crawl 怎么简单，怎么清楚就怎么来。这个爬虫只爬取当日电影名字，那我们只需要在网页中采集这一项即可。

1．选择爬取的项目 items.py

修改 items.py 文件如下：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class TodaymoiveItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    moiveName = scrapy.Field()  #电影名字
```



由于 Python 中严格的格式检查。Python 中最常见的异常 IndentationError 会经常出现。如果使用的编辑器是 vi 或者 vim，强烈建议修改 vi 的全局配置文件/etc/vim/vimrc，将所有的 4 个空格变成 tab。

与最初的 items.py 比较一下，修改后的文件只是按照原文的提示添加了需要爬取的项目，然后将类结尾的 pass 去掉了。这个类的定义不同于标准的 Python 类，它没有一般 Python 类的__init__的解析函数，没有类函数，只定义了类成员。虽然它从结构上更类似于 Python 的字典，但毫无疑问它的确就是一个 Python 类。

2．定义怎样爬取 wuHanMoiveSpider.py

修改 wuHanMoiveSpider.py，内容如下：

```
 1 # -*- coding: utf-8 -*-
 2 import scrapy
 3 from todayMoive.items import TodaymoiveItem
 4
 5
 6 class WuhanmoiveSpider(scrapy.Spider):
 7     name = "wuHanMoiveSpider"
 8     allowed_domains = ["jycinema.com"]
 9     start_urls = (
10         'http://www.jycinema.com/browsing/Cinemas/Details/1029',
11     )
12
13     def parse(self, response):
14         subSelector = response.xpath('//div[@class="film-header"]') #嵌套
选择
15         items = []
16         for sub in subSelector:
17             item = TodaymoiveItem()
18             item['moiveName'] = sub.xpath('./a/h3/text()').extract()
19             items.append(item)
20         return items
```

在这个 python 文件中，首先导入了 scrapy 模块，然后从模块（包）todayMoive 中的 items 文件中导入了 TodaymoiveItem 类，也就是刚才定义需要爬行内容的那个类。WuhanmoiveSpider 是一个自定义的爬虫类，它是由 scrapy genspider 命令自动生成的。这个自定义类继承于 scrapy.Spider 类。第 7 行的 name 定义的是爬虫名。第 8 行的 allowed_domains 定义的是域范围，也就是说该爬虫只能在这个域内爬行。第 9 行的 start_urls 定义的是爬行的网页，这个爬虫只需要爬行一个网页，所以在这里 start_urls 可以是一个元组类型。如果需要爬行多个网页，最好使用列表类型，以便于随时在后面添加需要爬行的网页。

爬虫类中的 parse 函数需要参数 response，这个 response 就是请求网页后返回的数据。至于怎么从 response 中选取所需的内容，笔者一般采取两种方法，一是直接在网页上查看网页源代码，二是自己写个 python 程序用 urllib2 将网页返回的内容写入到文本文件中，再慢慢地查询。

打开 Chrome 浏览器，在地址栏输入爬取网页的地址，打开网页，如图 5-12 所示。



图 5-12　爬虫来源网页

同一网页内的同一项目格式基本上都是相同的，所以只需要找到一个项目的选择器位置就可以了。就以最上面的这个电影《宝贝当家》为例，在网页中右击空白处，在弹出菜单中选择“查看网页源代码”，如图 5-13 所示。



图 5-13　查看网页源代码

打开源代码网页，按 Ctrl+F 组合键，在查找框中输入“宝贝当家”后按回车键，查找结果如图 5-14 所示。

```
宝贝当家                                第 1 条，共 1 条

<div class="film-list">
        <div class="film-item " data-cinema-id="1029" data-movie-id="dscaDdMXgL" data-attribute-short-name="">
            <div class="movie image">
                <div class="image-outer">
                    <div style="background-image: url('//www.jycinema.com/CDN/Image/Entity/FilmPosterGraphic/h-dscaDdMXgL?
width=121&amp;height=180')">
                        <img src="//www.jycinema.com/CDN/Image/Entity/FilmPosterGraphic/h-dscaDdMXgL?width=121&amp;height=180" />
                    </div>
                </div>
                <img class="rating-image" src="//www.jycinema.com/CDN/Image/Entity/RatingIconGraphic/G?width=22&amp;height=22" alt="" width="22"
height="22" />
            </div>
            <div class="film-showtimes">
                <div class="film-header">
                    <a href="//www.jycinema.com/browsing/Movies/Details/h-dscaDdMXgL">
                        <h3 class="film-title">宝贝当家 （数字）</h3>
                    </a>
                </div>
```

图 5-14　查找关键词

整个源代码网页只有一个查询结果，那就是它了。仔细看看怎样才能得到这个字符串

呢？用 XPath 一个一个地数 Tag 标签，不是不可以，只是使用 Scrapy 就是为了简单方便，如果要一个一个地数，那和使用 re 模块获取字符串也就没什么区别了。

这里通常使用嵌套选择器。一个字符串不好选，但这个标签块却很好选，然后在标签块中选择字符串，这样就方便多了。比如选择离目标字符串最近的标签<div class="film-header">，在脚本中执行语句：

```
        subSelector = response.xpath('//div[@class="film-header"]')
```

意思是选择整个网页中所有以 div 为标签 class 属性为“film-header”的块，subSelector 是一个列表，里面装的是选择器。然后再执行语句：

```
        for sub in subSelector:
            item = TodaymoiveItem()
            item['moiveName'] = sub.xpath('./a/h3/text()').extract()
```

意思是在这个块中，选择下级标签为<a>的下级标签为<h3>的字符串，并将其解析出来。选择器的选择到底对不对呢？可以验证一下，在该项目的任意一级目录下，执行命令：

```
scrapy shell http://www.jycinema.com/browsing/Cinemas/Details/1029
```

执行结果如图 5-15 所示。

```
king@debian: ~/code/crawler/scrapyProject/todayMoive
2016-07-26 23:34:33+0800 [scrapy] INFO: Enabled spider middlewares: HttpErrorMid
dleware, OffsiteMiddleware, RefererMiddleware, UrlLengthMiddleware, DepthMiddlew
are
2016-07-26 23:34:33+0800 [scrapy] INFO: Enabled item pipelines: TodaymoivePipeli
ne
2016-07-26 23:34:33+0800 [scrapy] DEBUG: Telnet console listening on 127.0.0.1:6
023
2016-07-26 23:34:33+0800 [scrapy] DEBUG: Web service listening on 127.0.0.1:6080
2016-07-26 23:34:33+0800 [wuHanMoiveSpider] INFO: Spider opened
2016-07-26 23:34:33+0800 [wuHanMoiveSpider] DEBUG: Crawled (200) <GET http://www
.jycinema.com/browsing/Cinemas/Details/1029> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x7fb0ed57d910>
[s]   item       {}
[s]   request    <GET http://www.jycinema.com/browsing/Cinemas/Details/1029>
[s]   response   <200 http://www.jycinema.com/browsing/Cinemas/Details/1029>
[s]   settings   <scrapy.settings.Settings object at 0x7fb0edf76750>
[s]   spider     <WuhanmoiveSpider 'wuHanMoiveSpider' at 0x7fb0ec715110>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

In [1]:
```

图 5-15　scrapy shell

response 后面的 200 是网页返回代码，200 代表获取数据正常返回，如果出现其他的数字，那就得仔细检查代码了。现在可以放心地验证了，执行命令：

```
subSelector = response.xpath('//div[@class="film-header"]')
print subSelector[0].xpath('./a/h3/text()').extract()
print subSelector[0].xpath('./a/h3/text()').extract()[0].encode('utf8')
```

执行结果如图 5-16 所示。

```
king@debian: ~/code/crawler/scrapyProject/todayMoive
2016-07-26 23:42:37+0800 [wuHanMoiveSpider] INFO: Spider opened
2016-07-26 23:42:38+0800 [wuHanMoiveSpider] DEBUG: Crawled (200) <GET http://www
.jycinema.com/browsing/Cinemas/Details/1029> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x7f776fad0910>
[s]   item       {}
[s]   request    <GET http://www.jycinema.com/browsing/Cinemas/Details/1029>
[s]   response   <200 http://www.jycinema.com/browsing/Cinemas/Details/1029>
[s]   settings   <scrapy.settings.Settings object at 0x7f77704c9750>
[s]   spider     <WuhanmoiveSpider 'wuHanMoiveSpider' at 0x7f776ec68110>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

In [1]: subSelector = response.xpath('//div[@class="film-header"]')

In [2]: print subSelector[0].xpath('./a/h3/text()').extract()
[u'\u5b9d\u8d1d\u5f53\u5bb6 \uff08\u6570\u5b57\uff09']

In [3]: print subSelector[0].xpath('./a/h3/text()').extract()[0].encode('utf8')
宝贝当家 （数字）

In [4]: 
```

图 5-16　验证选择器

第一次 print 的结果是一堆数字码，那是因为 scrapy 默认将爬取的内容转成了 unicode 码。如果要把它变成可见的汉字，那就将字符串使用 encode('utf8')转换成 utf8 码。

看来选择器的选择没问题。再回头看看 wuHanMoiveSpider.py 中的 parse 函数就很容易理解了。代码第 14 行先用选择器选择了一个“块”。第 15 行定义了一个 items 的空列表，这里定义 items 的列表是因为返回的 item 不止一个，所以只能让 item 以列表的形式返回。第 17 行 item 初始化为一个 TodaymoizeItem()的类，这个类是从 todayMoive.items 中导入过来的。第 18 行将已经初始化类 item 中的 moiveName 项赋值。第 19 行将 item 追加到 items 列表中去。最后 return items，注意这里返回的是 items，不是 item。

3．保存爬取的结果 pipelines.py

修改 pipelines.py，内容如下：

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

import time

class TodaymoivePipeline(object):
    def process_item(self, item, spider):
        now = time.strftime('%Y-%m-%d', time.localtime())
        fileName = 'wuHan' + now + '.txt'
        with open(fileName,'a') as fp:
            fp.write(item['moiveName'][0].encode('utf8') + '\n\n')
        return item
```

这个脚本没什么可说的，比较简单。就是把当日的年月日抽取出来当成文件名的一部分。然后把 wuHanMoiveSpider.py 中获取项的内容输入到该文件中。这个脚本中只需要注意两点。第一，open 函数创建文件时必须是以追加的形式创建，也就是说 open 函数的第二个参数必须是 a。因为 wuHanMoiveSpider.py 返回的是一个 item 列表 items，这里的写入文件只能一个一个 item 地写入。如果 open 函数的第二个参数是 w，造成的后果就是先擦除前面写入的内容，再写入新内容，一直循环到 items 列表结束，最终的结果就是文件里只保存了最后一个 item 的内容。第二是保存文件中的内容如果含有汉字就必须转换成 utf8 码。汉字的 unicode 码保存到文件中正常人类都是无法识别的，所以还是转换成正常人类能识别的 utf8 吧。

到了这一步，这个 Scrapy 爬虫基本上完成了。回到 scrapy.cfg 文件的同级目录下（实际上只要是在 todayMoive 项目下的任意目录中执行都行，之所以在这一级目录执行纯粹是为了美观而已），执行命令：

```
scrapy crawl wuHanMoiveSpider
```

结果却什么都没有？为什么呢？

4．分派任务的 settings.py

先看看 settings.py 的初始代码。它仅指定了 Spider 爬虫的位置。再看看写好的 Spider 爬虫的开头，它导入了 items.py 作为模块，也就是说现在 Scrapy 已经知道了爬取哪些项目，怎样爬取内容，而 pipelines 说明了最终的爬取结果怎样处理。唯一不知道的就是由谁来处理这个爬行结果，这时候就该 setting.py 出点力气了。setting.py 的最终代码如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for todayMoive project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'todayMoive'

SPIDER_MODULES = ['todayMoive.spiders']
NEWSPIDER_MODULE = 'todayMoive.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'todayMoive (+http://www.yourdomain.com)'
```

```

### user define
ITEM_PIPELINES = {'todayMoive.pipelines.TodaymoivePipeline':300}
```

这跟初始的 settings.py 相比，就是在最后添加了一行 ITEM_PIPELINES。它告诉 Scrapy 最终的结果是由 todayMoive 模块中 pipelines 模块的 TodaymoivePipeline 类来处理。ITEM_PIPELINES 是一个字典，字典的 key 用来处理结果的类，字典的 value 是这个类执行的顺序。这里只有一种处理方式，value 填多少都没问题。如果需要多种处理结果的方法，那就要确立顺序了。数字越小的越先被执行。

现在可以测试这个 Scrapy 爬虫了，还是执行命令：

```
scrapy crawl wuHanMoiveSpider
ls
more wuhan2016-07-27.txt
```

执行结果如图 5-17 所示。



图 5-17　Scrapy 爬虫结果

好了，这个最简单的爬虫就到这里了。从这个项目可以看出，Scrapy 爬虫只需要顺着思路照章填空就可以了。如果需要的项比较多，获取内容的网页源比较复杂或者不规范，可能会稍微麻烦点，但处理起来基本上都是大同小异的。与前章的 re 爬虫相比，越复杂的爬虫就越能体现 Scrapy 的优势。

5.4 Scrapy 爬虫实战二：天气预报

上节使用 Scrapy 做了一个最简单的爬虫。本节稍微增加点难度，做个所需项目多一点的爬虫，并将爬虫的结果以多种形式保存起来。我们就从网络天气预报开始吧。

5.4.1 项目准备

首先要做的是确定网络天气数据的来源。打开百度，搜索“网络天气预报”，搜索结果如图 5-18 所示。



图 5-18　百度搜索数据来源站点

有很多网站可以选择，任意选择一个都可以。这里笔者选择的是 http://wuhan.tianqi.com/。在浏览器中打开该网站，并找到所属的城市，将会出现当地一周的天气预报，如图 5-19 所示。



图 5-19　本地一周天气

在这里，包含的信息有城市日期、星期、天气图标、温度、天气状况以及风向。除了天气图标是以图片的形式显示，其他的几项都是字符串。本节 Scrapy 爬虫的目标将包含所有的有用信息。至此，items.py 文件已经呼之欲出了。

5.4.2　创建编辑 Scrapy 爬虫

首先还是打开 Putty，连接到 Linux。在工作目录下创建 Scrapy 项目，并根据提示依照 spider 基础模版创建一个 spider。执行命令：

```
cd
cd code/crawler/scrapyProject
scrapy startproject weather
cd weather
scrapy genspider wuHanSpider wuhan.tianqi.com
```

执行结果如图 5-20 所示。

```
king@debian: ~/code/crawler/scrapyProject/weather
king@debian:~$ cd
king@debian:~$ cd code/crawler/scrapyProject/
king@debian:~/code/crawler/scrapyProject$ scrapy startproject weather
New Scrapy project 'weather' created in:
    /mnt/disk/sync/code/crawler/scrapyProject/weather

You can start your first spider with:
    cd weather
    scrapy genspider example example.com
king@debian:~/code/crawler/scrapyProject$ cd weather
king@debian:~/code/crawler/scrapyProject/weather$ scrapy genspider wuHanSpider w
uhan.tianqi.com
Created spider 'wuHanSpider' using template 'basic' in module:
  weather.spiders.wuHanSpider
king@debian:~/code/crawler/scrapyProject/weather$
```

图 5-20　创建 Scrapy 项目

项目模版创建完毕，项目文件如图 5-21 所示。

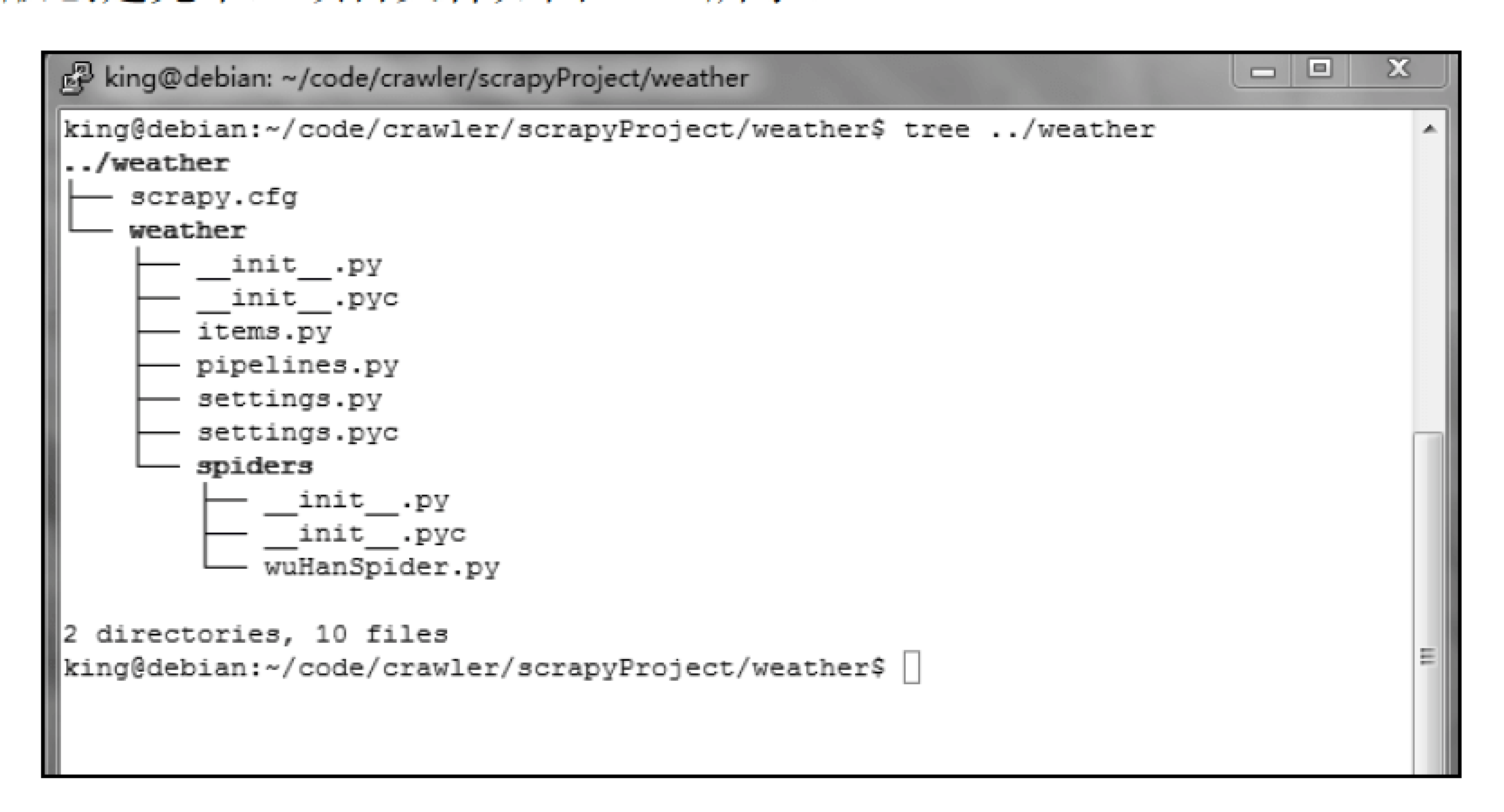


图 5-21　基础项目模版

1．修改 items.py

按照上一节中的顺序，第一个要修改的还是 items.py。修改后的 items.py 代码如下：

```
# -*- coding: utf-8 -*-

```

```
# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class WeatherItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    cityDate = scrapy.Field()
    week = scrapy.Field()
    img = scrapy.Field()
    temperature = scrapy.Field()
    weather = scrapy.Field()
    wind = scrapy.Field()
```

在 items.py 文件中，只需要将希望获取的项名称按照文件中示例的格式填入进去即可。唯一需要注意的就是每一行最前面的到底是空格还是 Tabstop。这个文件可以说是 Scrapy 爬虫中最没有技术含量的一个文件了。填空，就是填空而已。

2．修改 Spider 文件 wuHanSpider.py

按照上一节的顺序，第二个修改的文件应该轮到 spiders/wuHanSpider.py 了。暂时先不要修改文件，使用 scrapy shell 命令来测试、获取选择器。执行命令：

```
scrapy shell http://wuhan.tianqi.com
```

执行结果如图 5-22 所示。

```
king@debian: ~/code/crawler/scrapyProject/weather/weather
tMiddleware, CookiesMiddleware, ChunkedTransferMiddleware, DownloaderStats
2016-07-28 02:13:53+0800 [scrapy] INFO: Enabled spider middlewares: HttpErrorMid
dleware, OffsiteMiddleware, RefererMiddleware, UrlLengthMiddleware, DepthMiddlew
are
2016-07-28 02:13:53+0800 [scrapy] INFO: Enabled item pipelines:
2016-07-28 02:13:53+0800 [scrapy] DEBUG: Telnet console listening on 127.0.0.1:6
023
2016-07-28 02:13:53+0800 [scrapy] DEBUG: Web service listening on 127.0.0.1:6080
2016-07-28 02:13:53+0800 [wuHanSpider] INFO: Spider opened
2016-07-28 02:13:53+0800 [wuHanSpider] DEBUG: Crawled (200) <GET http://wuhan.ti
anqi.com> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x7f6403811950>
[s]   item       {}
[s]   request    <GET http://wuhan.tianqi.com>
[s]   response   <200 http://wuhan.tianqi.com>
[s]   settings   <scrapy.settings.Settings object at 0x7f640420a790>
[s]   spider     <WuhanspiderSpider 'wuHanSpider' at 0x7f64029a9050>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

In [1]:
```

图 5-22　scrapy shell

从上图可看出 response 的返回代码为 200，是正常返回，已成功获取该网页的 response。下面开始试验选择器了。打开 Chrome 浏览器（任意一个浏览器都可以，哪个方便用哪个），在地址栏输入 http://wuhan.tianqi.com，按 Enter 键打开网页。在“武汉今天天气”这个框架内任意空白处单击右键，选择“查看框架源代码”（有的城市页面没有使用框架如上海，那还是选择“查看网页源代码”），如图 5-23 所示。



图 5-23　查看源代码

在框架源代码页，使用 Ctrl+f 组合键查找关键词“星期”，很容易就找到所需数据的位置，如图 5-24 所示。

```
            <span style="line-height:32px;" class="zhoubiantianqi"><a href="http://wuha
href="http://wuhan.tianqi.com/15/">武汉15天天气</a>  <a href="http://wuhan.ti
          </div>
          <div class="sixday_detail">
            <div class="everytqshow" id="detail">
              <div class="tqshow1"><h3>武汉<font color="#0066cc">今日</font>天气</h3><p>星期五</p><ul><li class="tqpng"><img class='
align='absmiddle' src='http://img.tianqi.com/static/images/tianqibig/b0.png' style='border:0;width:46px;height:46px'/></li><li><font
color="#f00">37℃</font>~<font color="#4899be">27℃</font></li><li>晴</li><li style="height:18px;overflow:hidden">南风 1级</li></ul><
              <div class="tqshow1"><h3>武汉<font color="red">明日</font>天气</h3><p><font color='green'>星期六</font></p><ul><li
class="tqpng"><img class='pngtqico' align='absmiddle' src='http://img.tianqi.com/static/images/tianqibig/b0.png'
style='border:0;width:46px;height:46px'/></li><li><font color="#f00">37℃</font>~<font color="#4899be">27℃</font></li><li>晴</li><li
style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
              <div class="tqshow1"><h3>武汉<font color="red">后天</font>天气</h3><p><font color='green'>星期日</font></p><ul><li
class="tqpng"><img class='pngtqico' align='absmiddle' src='http://img.tianqi.com/static/images/tianqibig/b0.png'
style='border:0;width:46px;height:46px'/></li><li><font color="#f00">37℃</font>~<font color="#4899be">27℃</font></li><li>晴</li><li
style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
              <div class="tqshow1"><h3>武汉01日天气</h3><p>星期一</p><ul><li class="tqpng"><img class='pngtqico' align='absmiddle'
src='http://img.tianqi.com/static/images/tianqibig/b1.png' style='border:0;width:46px;height:46px'/></li><li><font color="#f00">35℃<
<font color="#4899be">27℃</font></li><li>多云</li><li style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
              <div class="tqshow1"><h3>武汉02日天气</h3><p>星期二</p><ul><li class="tqpng"><img class='pngtqico' align='absmiddle'
src='http://img.tianqi.com/static/images/tianqibig/b7.png' style='border:0;width:46px;height:46px'/></li><li><font color="#f00">35℃<
<font color="#4899be">27℃</font></li><li>小雨</li><li style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
              <div class="tqshow1"><h3>武汉03日天气</h3><p>星期三</p><ul><li class="tqpng"><img class='pngtqico' align='absmiddle'
src='http://img.tianqi.com/static/images/tianqibig/b7.png' style='border:0;width:46px;height:46px'/></li><li><font color="#f00">34℃<
<font color="#4899be">26℃</font></li><li>小雨</li><li style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
            </div>
          </div>
```

星期　　第 1 条，共 7 条

图 5-24　查找所需数据位置

因为所需数据都是以<div class="tqshow1">为开头的，试下查找页面还有没有其他以<div class="tqshow1">为开头的数据。如果没有就可以将<div class="tqshow1">作为 XPath 的锚点，如图 5-25 所示。

```
            <span style="line-height:32px;" class="zhoubiantianqi"><a href="http://wuhar
="http://wuhan.tianqi.com/15/">武汉15天天气</a>  <a href="http://wuhan.tia
        </div>
        <div class="sixday_detail">
          <div class="everytqshow" id="detail">
            <div class="tqshow1"><h3>武汉<font color="#0066cc">今日</font>天气</h3><p>星期五</p><ul><li class="tqpng"><img class=
n=' absmiddle' src='http://img.tianqi.com/static/images/tianqibig/b0.png' style='border:0;width:46px;height:46px'/></li><li><font
r="#f00">37℃</font>~<font color="#4899be">27℃</font></li><li>晴</li><li style="height:18px;overflow:hidden">南风 1级</li></ul><
            <div class="tqshow1"><h3>武汉<font color="red">明日</font>天气</h3><p><font color='green'>星期六</font></p><ul><li
s="tqpng"><img class='pngtqico' align='absmiddle' src='http://img.tianqi.com/static/images/tianqibig/b0.png'
e='border:0;width:46px;height:46px'/></li><li><font color="#f00">37℃</font>~<font color="#4899be">27℃</font></li><li>晴</li><li
e="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
            <div class="tqshow1"><h3>武汉<font color="red">后天</font>天气</h3><p><font color='green'>星期日</font></p><ul><li
s="tqpng"><img class='pngtqico' align='absmiddle' src='http://img.tianqi.com/static/images/tianqibig/b0.png'
e='border:0;width:46px;height:46px'/></li><li><font color="#f00">37℃</font>~<font color="#4899be">27℃</font></li><li>晴</li><li
e="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
            <div class="tqshow1"><h3>武汉01日天气</h3><p>星期一</p><ul><li class="tqpng"><img class='pngtqico' align='absmiddle'
'http://img.tianqi.com/static/images/tianqibig/b1.png' style='border:0;width:46px;height:46px'/></li><li><font color="#f00">35℃<
t color="#4899be">27℃</font></li><li>多云</li><li style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
            <div class="tqshow1"><h3>武汉02日天气</h3><p>星期二</p><ul><li class="tqpng"><img class='pngtqico' align='absmiddle'
'http://img.tianqi.com/static/images/tianqibig/b7.png' style='border:0;width:46px;height:46px'/></li><li><font color="#f00">35℃<
t color="#4899be">27℃</font></li><li>小雨</li><li style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
            <div class="tqshow1"><h3>武汉03日天气</h3><p>星期三</p><ul><li class="tqpng"><img class='pngtqico' align='absmiddle'
'http://img.tianqi.com/static/images/tianqibig/b7.png' style='border:0;width:46px;height:46px'/></li><li><font color="#f00">34℃<
t color="#4899be">26℃</font></li><li>小雨</li><li style="height:18px;overflow:hidden">无持续风向 微风</li></ul></div>
          </div>
        </div>
```

<div class="tqshow1">　第 1 条，共 6 条

图 5-25　测试锚点

从测试结果看出，整个源代码页面总共只有 6 个<div class="tqshow1">标签，可以将它作为 XPath 锚点。回到 Putty 下的 scrapy shell 中，执行命令：

```
subSelector = response.xpath('//div[@class="tqshow1"]')
subSelector
```

执行结果如图 5-26 所示。



图 5-26　确定 XPath 锚点

然后从 subSelector 中提取有效数据，如图 5-27 所示。

```
<div class="tqshow1"><h3>武汉<font color="#0066cc">今日</font>天气</h3><p>星期五</p><ul><li class="tqpn
' src='http://img.tianqi.com/static/images/tianqibig/b0.png' style='border:0;width:46px;height:46px'/><
C</font>~<font color="#4899be">27℃</font></li><li>晴</li><li style="height:18px;overflow:hidden">南风
```

king@debian: ~/code/crawler/scrapyProject

```
In [18]: subSelector[0].xpath('./h3//text()').extract()
Out[18]: [u'\u6b66\u6c49', u'\u4eca\u65e5', u'\u5929\u6c14']

In [19]: subSelector[0].xpath('./p/text()').extract()
Out[19]: [u'\u661f\u671f\u4e94']

In [20]: subSelector[0].xpath('./ul/li[1]/img/@src').extract()
Out[20]: [u'http://img.tianqi.com/static/images/tianqibig/b0.png']

In [21]: subSelector[0].xpath('./ul/li[2]//text()').extract()
Out[21]: [u'37\u2103', u'~', u'27\u2103']

In [22]: subSelector[0].xpath('./ul/li[3]//text()').extract()
Out[22]: [u'\u6674']

In [23]: subSelector[0].xpath('./ul/li[4]//text()').extract()
Out[23]: [u'\u5357\u98ce 1\u7ea7']
```

图 5-27　XPath 选择器获取数据

图 5-27 中标识了 item['cityDates']所需数据的来源位置和 XPath 选择器的工作流程（因为 h3 标签内还含有其他的标签，所以这里选择方式为'.h3//text()'而不是'.h3/text()'）。到这一步，Scrapy 项目的 Spider 文件 wuHanSpider.py 也已成形了。wuHanSpider.py 的代码如下：

```
# -*- coding: utf-8 -*-
import scrapy
from weather.items import WeatherItem

class WuhanspiderSpider(scrapy.Spider):
    name = "wuHanSpider"
    allowed_domains = ["tianqi.com"]
    citys = ['wuhan','shanghai']
    start_urls = []
    for city in citys:
        start_urls.append('http://' + city + '.tianqi.com/')

    def parse(self, response):
        subSelector = response.xpath('//div[@class="tqshow1"]')
        items = []
        for sub in subSelector:
            item = WeatherItem()
            cityDates = ''
            for cityDate in sub.xpath('./h3//text()').extract():
                cityDates += cityDate
            item['cityDate'] = cityDates
            item['week'] = sub.xpath('./p//text()').extract()[0]
            item['img'] = sub.xpath('./ul/li[1]/img/@src').extract()[0]
            temps = ''
            for temp in sub.xpath('./ul/li[2]//text()').extract():
```

```
26                 temps += temp
27             item['temperature'] = temps
28              item['weather'] = sub.xpath('./ul/li[3]//text()').extract()[0]
29             item['wind'] = sub.xpath('./ul/li[4]//text()').extract()[0]
30             items.append(item)
31         return items
```

文件开头别忘了导入 scrapy 模块和 items 模块。在第 8~11 行中，给 start_urls 列表添加了上海天气的网页（刚创建时 wuHanSpider.py 时，start_urls 是一个元组，为了便于添加多个网页，所以将 start_urls 改成了列表）。如果还想添加其他的城市天气，可以在第 8 行的 citys 列表中添加城市代码。

3．修改 pipelines.py，处理 Spider 的结果

这里还是将 Spider 的结果保存为 txt 格式，以便于阅读。pipelines.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

import time
import os.path
import urllib2

class WeatherPipeline(object):
    def process_item(self, item, spider):
        today = time.strftime('%Y%m%d', time.localtime())
        fileName = today + '.txt'
        with open(fileName,'a') as fp:
            fp.write(item['cityDate'].encode('utf8') + '\t')
            fp.write(item['week'].encode('utf8') + '\t')
            imgName = os.path.basename(item['img'])
            fp.write(imgName + '\t')
            if os.path.exists(imgName):
                pass
            else:
                with open(imgName, 'wb') as fp:
                    response = urllib2.urlopen(item['img'])
                    fp.write(response.read())
            fp.write(item['temperature'].encode('utf8') + '\t')
            fp.write(item['weather'].encode('utf8') + '\t')
```

```
29            fp.write(item['wind'].encode('utf8') + '\n\n')
30            time.sleep(1)
31        return item
```

第 1 行，确认字符编码。回想一下，在 Spider 文件 wuHanSpider.py 中似乎也有这一行。这是因为 Scrapy 默认将所有获取数据的编码定义成 utf8。第 8~10 行，导入所需的模块。第 14~15 行，用 time 模块确定了当天的年月日，并将其作为文件名。第 17~29 行，将所获取的数据转换为编码，然后存储到文件中。第 30 行用 time.sleep 暂停 1 秒，可以避免因为数据写入太快而丢失数据的问题。如果得到的数据为图片，将使用 urllib2 模块下载图片到当前目录下。os 模块用于判断图片是否已经下载过。

4．修改 settings.py，决定由哪个文件来处理获取的数据

这个就很简单了，跟上节的 settings.py 没什么区别，直接将 pipelines.py 添加到 ITEM_PIPELINES 中去就可以了。settings.sp 文件内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for weather project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'weather'

SPIDER_MODULES = ['weather.spiders']
NEWSPIDER_MODULE = 'weather.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'weather (+http://www.yourdomain.com)'

#### user add
ITEM_PIPELINES = {
'weather.pipelines.WeatherPipeline':1,
}
```

最后，回到 weather 项目下，执行命令：

```
scrapy crawl wuHanSpider
more *.txt
```

得到结果如图 5-28 所示。

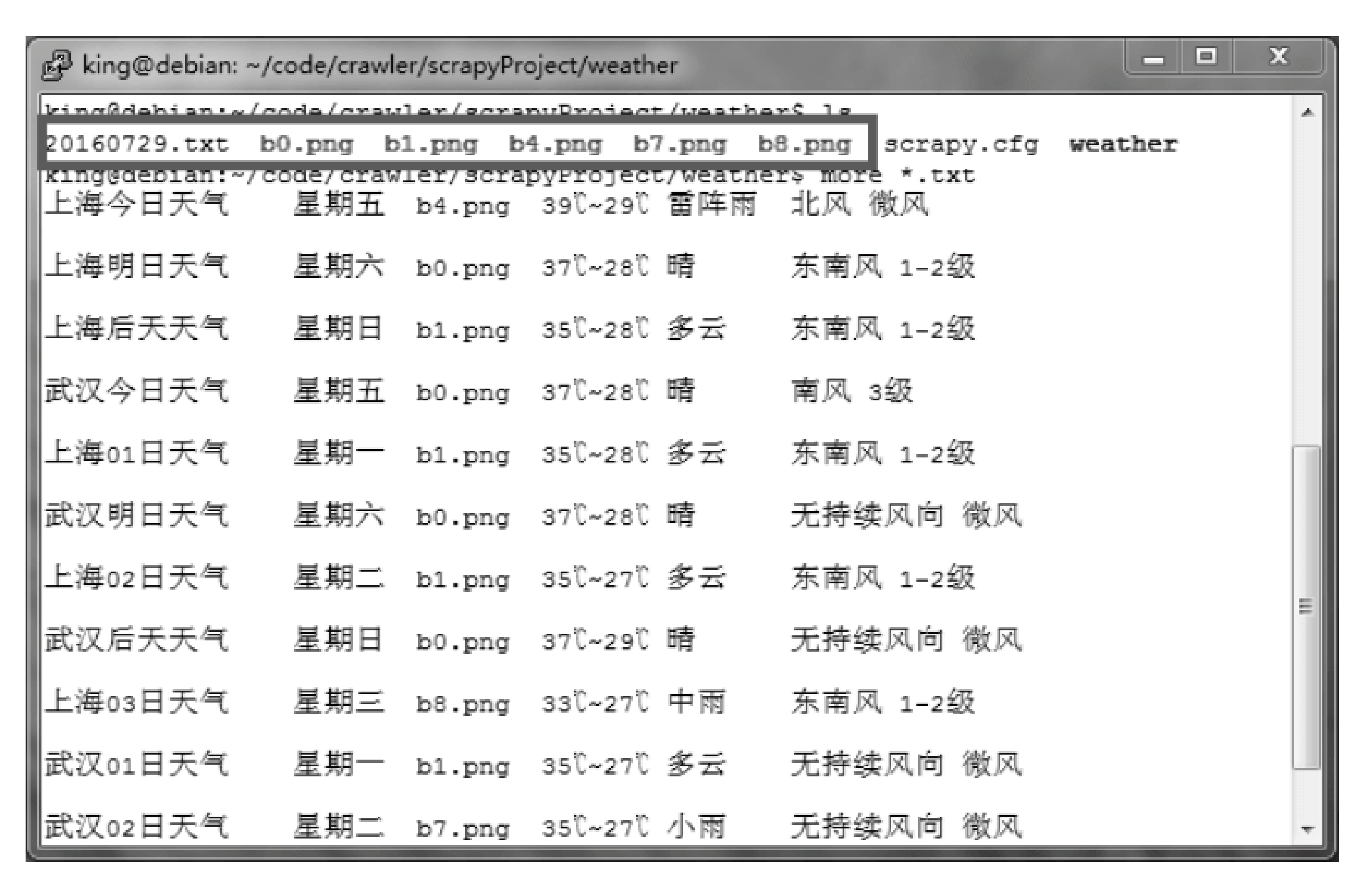


图 5-28　保存结果为 txt

至此，一个完整的 Scrapy 爬虫已经完成了。这个爬取天气的爬虫比上一个爬取电影的爬虫稍微复杂一点，但流程基本是一样的，都是做填空题而已。

5.4.3　数据存储到 json

上节已经完成了一个 Scrapy 爬虫，并将其爬取的结果保存到了 txt 文件。但 txt 文件的优点仅仅是方便阅读，而程序阅读一般都是使用更方便的 json、cvs 等等格式。有时程序员更加希望将爬取的结果保存到数据库中便于分析统计。所以，本节将继续讲解 Scrapy 爬虫的保存方式，也就是继续对 pipelines.py 动手术。

这里以 json 格式为例，其他的格式都大同小异，读者可自行摸索测试。既然是保存为 json 格式，当然就少不了 Python 的 json 模块了。幸运的是 json 模块是 Python 的标准模块，无须安装可直接使用。

保存爬取结果，那必定涉及了 pipelines.py。我们可以直接修改这个文件，然后再修改一下 settings.py 中的 ITEM_PIPELINES 项即可。但是仔细看看 settings.py 中的 ITEM_PIPELINES 项，它是一个字典。字典是可以添加元素的。因此完全可以自行构造一个 Python 文件，然后把这个文件添加到 ITEM_PIPELINES 不就可以了吗？这个思路是否可行，测试一下就知道了。

为了“表明身份”，笔者给这个新创建的 Python 文件取名为 pipelines2json.py，这个名字简单明了，而且显示了与 pipellines.py 的关系。pipelines2.json 的文件内容如下：

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
```

```
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

import time
import json
import codecs

class WeatherPipeline(object):
    def process_item(self, item, spider):
        today = time.strftime('%Y%m%d', time.localtime())
        fileName = today + '.json'
        with codecs.open(fileName, 'a', encoding='utf8') as fp:
            line = json.dumps(dict(item), ensure_ascii=False) + '\n'
            fp.write(line)
        return item
```

然后修改 settings.py 文件，将 pipelines2json 加入到 ITEM_PIPELINES 中去。修改后的 settings.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for weather project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'weather'

SPIDER_MODULES = ['weather.spiders']
NEWSPIDER_MODULE = 'weather.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'weather (+http://www.yourdomain.com)'

#### user add
ITEM_PIPELINES = {
'weather.pipelines.WeatherPipeline':1,
'weather.pipelines2json.WeatherPipeline':2
}
```

测试一下效果。回到 weather 项目下执行命令：

```
scrapy crawl wuHanSpider
ls
more *.json
```

得到的结果如图 5-29 所示。

```
king@debian: ~/code/crawler/scrapyProject/weather
king@debian:~/code/crawler/scrapyProject/weather$ ls
20160729.json  b0.png  b4.png  b8.png      weather
20160729.txt   b1.png  b7.png  scrapy.cfg
king@debian:~/code/crawler/scrapyProject/weather$ more *.json
{"week": "星期五", "temperature": "37℃~28℃", "img": "http://img.tianqi.com/stati
c/images/tianqibig/b0.png", "cityDate": "武汉今日天气", "weather": "晴", "wind":
 "南风 3级"}
{"week": "星期六", "temperature": "37℃~28℃", "img": "http://img.tianqi.com/stati
c/images/tianqibig/b0.png", "cityDate": "武汉明日天气", "weather": "晴", "wind":
 "无持续风向 微风"}
{"week": "星期五", "temperature": "39℃~29℃", "img": "http://img.tianqi.com/stati
c/images/tianqibig/b4.png", "cityDate": "上海今日天气", "weather": "雷阵雨", "wi
nd": "北风 微风"}
{"week": "星期日", "temperature": "37℃~29℃", "img": "http://img.tianqi.com/stati
c/images/tianqibig/b0.png", "cityDate": "武汉后天天气", "weather": "晴", "wind":
 "无持续风向 微风"}
{"week": "星期六", "temperature": "37℃~28℃", "img": "http://img.tianqi.com/stati
c/images/tianqibig/b0.png", "cityDate": "上海明日天气", "weather": "晴", "wind":
 "东南风 1-2级"}
{"week": "星期一", "temperature": "35℃~27℃", "img": "http://img.tianqi.com/stati
c/images/tianqibig/b1.png", "cityDate": "武汉01日天气", "weather": "多云", "wind
": "无持续风向 微风"}
{"week": "星期日", "temperature": "35℃~28℃", "img": "http://img.tianqi.com/stati
c/images/tianqibig/b1.png", "cityDate": "上海后天天气", "weather": "多云", "wind
```

图 5-29 保存结果为 json

从上图看来试验成功了。按照这个思路，如果要将结果保存成 cvs 等格式，settings.py 应该怎么修改就很明显了。

5.4.4 数据存储到 MySQL

数据库有很多，MySQL、Sqlite3、Access、Postgresql 等等，可选择的范围很广。笔者选择的标准是，Python 支持良好、能够跨平台、使用方便，其中 Python 标准库默认支持 sqlite3。但谁让 Sqllit3 声名不显呢，Access 不能跨平台。所以这里笔者选择名气最大，Python 支持也不错的 MySQL。MySQL 使用人数众多，资料随处可见，出现问题咨询也挺方便。就是它了。

在 Linux 上安装 MySQL 很方便。首先连接 Putty 后，使用 root 用户权限，执行命令：

```
apt-get install mysql-server mysql-client
```

在安装过程中，会要求输入 MySQL 用户 root 的密码（此 root 非彼 root，一个是系统用户 root，一个是 MySQL 的用户 root）。

MySQL 安装完毕后，默认是自动启动的。首先连接到 MySQL 上，查看 MySQL 的字符编码。执行命令：

```
mysql -u root -p
SHOW VARIABLES LIKE “character%”;
```

执行结果如图 5-30 所示。

```
king@debian: ~
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW VARIABLES LIKE 'character%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | latin1                     |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | latin1                     |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)

mysql>
```

图 5-30　MySQL 默认字符编码

其中 character_set_database 和 character_set_server 设置的是 latin1 编码，刚才用 Scrapy 采集的数据都是 utf8 编码。如果直接将数据加入数据库必定会在编程处理中出现乱码问题，所以要稍微修改一下。网上流传着很多彻底修改 MySQL 默认字符编码的帖子，但由于版本的问题，不能通用。所以只能采取笨方法，不修改 MySQL 的环境变量，只在创建数据库和表的时候指定字符编码。创建数据库和表格，在 MySQL 环境下执行命令：

```
CREATE DATABASE scrapyDB CHARACTER SET 'utf8' COLLATE 'utf8_general_Ci';
USE scrapyDB;
CREATE TABLE weather(
id INT AUTO_INCREMENT,
cityDate char(24), week char(6),
img char(20),
temperature char(12),
weather char(20),
wind char(20),
PRIMARY KEY(id) )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

执行结果如图 5-31 所示。

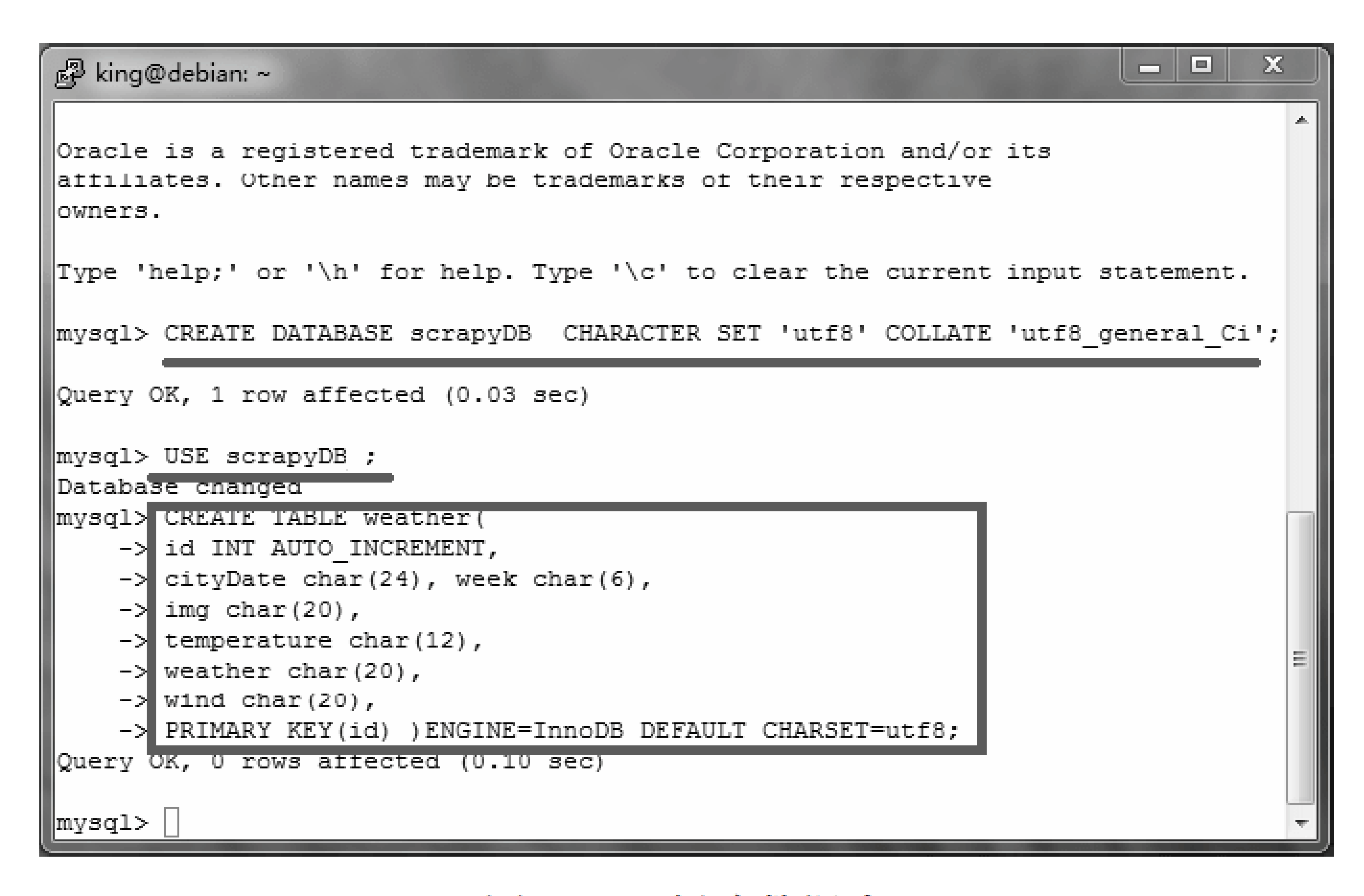


图 5-31　创建数据库

其中第一条命令是创建了一个默认字符编码为 utf8、名字为 scrapyDB 的数据库。第二条命令进入数据库。第三条命令是创建了一个默认字符编码为 utf8、名字为 wetaher 的表格。查看这个表格的结构，如图 5-32 所示。

```
king@debian: ~
mysql> show columns from weather;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| id          | int(11)  | NO   | PRI | NULL    | auto_increment |
| cityDate    | char(24) | YES  |     | NULL    |                |
| week        | char(6)  | YES  |     | NULL    |                |
| img         | char(20) | YES  |     | NULL    |                |
| temperature | char(12) | YES  |     | NULL    |                |
| weather     | char(20) | YES  |     | NULL    |                |
| wind        | char(20) | YES  |     | NULL    |                |
+-------------+----------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql>
```

图 5-32　查询表结构

由图 5-32 可以看出表格中的项基本与 wuHanSpider 爬取的项相同。至于多出来的那一项 id，是作为主键存在的。MySQL 的主键是不可重复的，而 wuHanSpider 爬取的项中没有符合这个条件的，所以还需要另外提供一个主键给表格更加合适。

创建完数据库和表格，下一步创建一个普通用户，并给普通用户管理数据库的权限。在 MySQL 环境下，执行命令：

```
    INSERT INTO mysql.user(Host,User,Password)
VALUES("%","crawlUSER",password("crawl123"));
    INSERT INTO mysql.user(Host,User,Password)
VALUES("localhost","crawlUSER",password("crawl123"));
    GRANT all privileges ON scrapyDB.* to crawlUSER@all IDENTIFIED BY
'crawl123';
    GRANT all privileges ON scrapyDB.* to crawlUSER@localhost IDENTIFIED BY
'crawl123';
```

执行结果如图 5-33 所示。

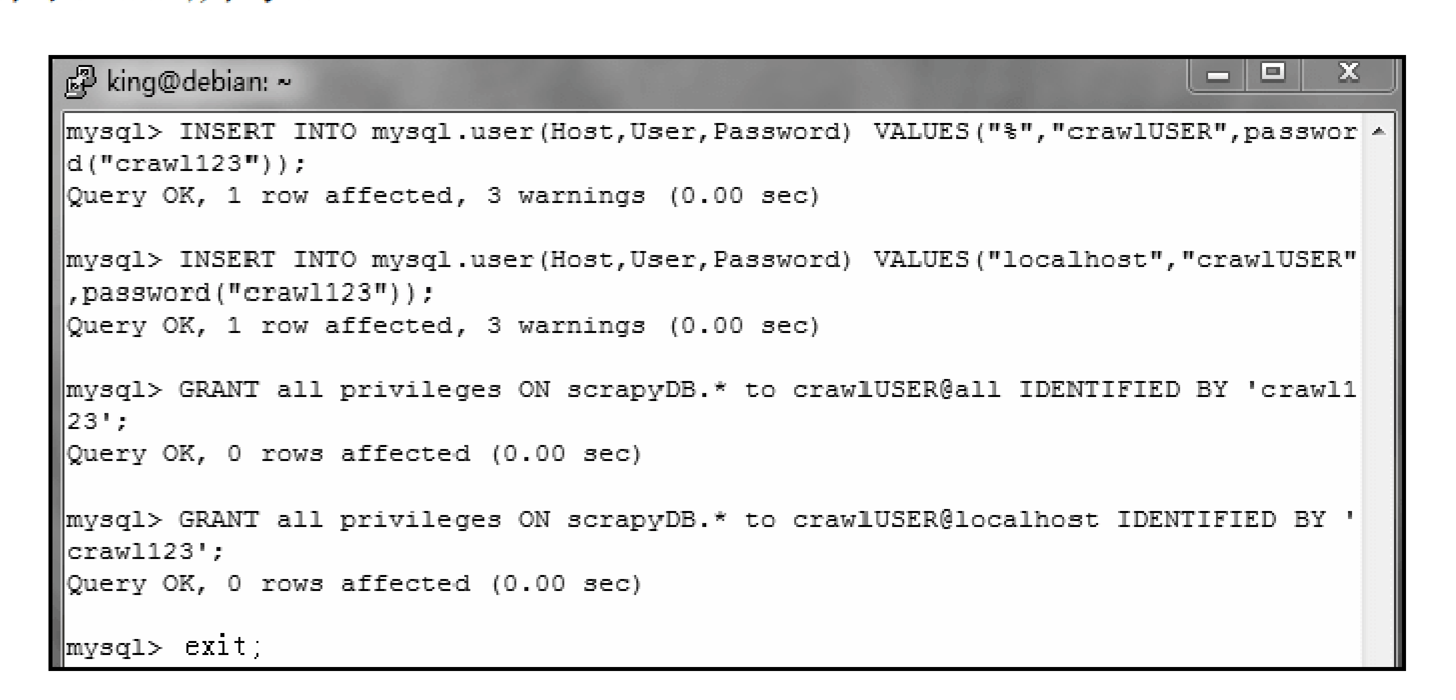


图 5-33　创建新用户、赋予管理权限

第 1 条命令创建了一个用户名为 crawlUSER 的远程用户，该用户只能远程登录，不能本

地登录。第 2 条命令创建了一个用户名为 crawlUSER 的本地用户，该用户只能本地登录，不能远程登录。第 3~4 条命令则赋予了 crawlUSER 用户管理 scrapyDB 数据库的所有权限。最后退出 MySQL。至此，数据库方面的配置已经完成，静待 Scrapy 来连接了。

Python 的标准库中没有直接支持 MySQL 的模块。在 Python 第三方库中能连接 MySQL 的不少，这里笔者选择使用最广的 MySQLdb 模块。

- Linux 中安装 MySQLdb 模块

在 Linux 下安装 MySQLdb 模块，最简单的方法是借助 Debian 庞大的软件库（可以说，只要不是私有软件，Debian 软件库总不会让人失望）。在终端下执行命令：

```
apt-get install python-mysqldb
```

执行结果如图 5-34 所示。

```
king@debian: ~
root@debian:/home/king# apt-get install python-mysqldb
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
python-mysqldb 已经是最新的版本。
python-mysqldb 被设置为手动安装。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 0 个软件包未被升级
。
root@debian:/home/king#
```

图 5-34　Linux 安装 MySQLdb 模块

系统已经安装过的会显示安装信息，没有安装的会自动安装。当然也可以使用 pip 安装。

- Windows 中安装 MySQLdb 模块

在 Windows 或 Mac OS 下安装第三方模块只能使用 pip，没有其他的选项。打开 cmd.exe，执行命令：

```
pip install MySQL-python
```

执行结果如图 5-35 所示。

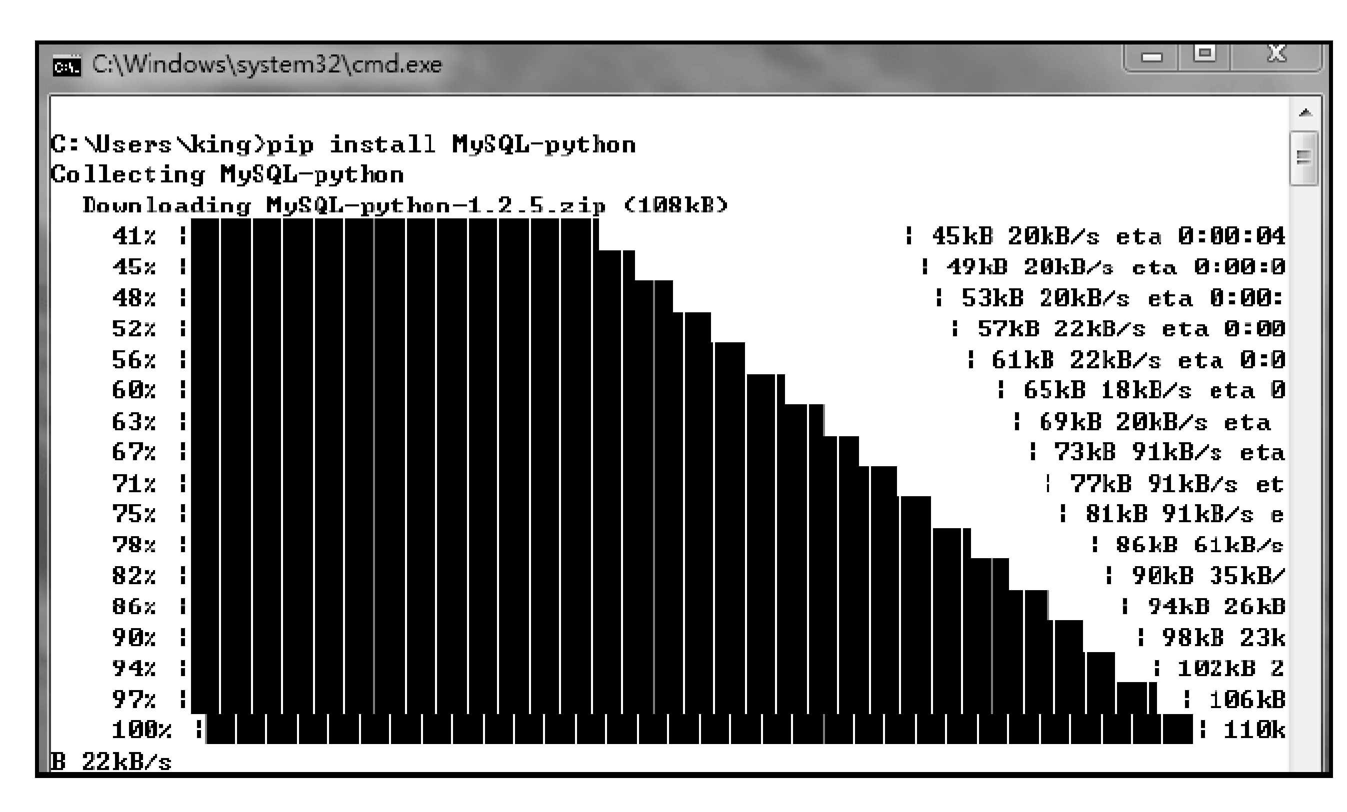


图 5-35　Windows 安装 MySQLdb 模块

这里只需要注意大小写即可。安装完毕后就可以将 MySQLdb 模块导入使用了。

Python 模块已经准备完毕，MySQL 的库表格也准备完毕，现在可以编辑 pipelines2mysql.py 了。在项目名为 weaterh 的 Scrapy 项目中的 pipelines.py 同层目录下，使用文本编辑器编写 pipelines2mysql.py，编辑完毕的 pipeliens2mysql.py 的内容如下：

```
1 # -*- coding: utf-8 -*-
2
3 # Define your item pipelines here
4 #
5 # Don't forget to add your pipeline to the ITEM_PIPELINES setting
6 # See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html
7
8 import MySQLdb
9 import os.path
10
11 class WeatherPipeline(object):
12     def process_item(self, item, spider):
13         cityDate = item['cityDate'].encode('utf8')
14         week = item['week'].encode('utf8')
15         img = os.path.basename(item['img'])
16         temperature = item['temperature'].encode('utf8')
17         weather = item['weather'].encode('utf8')
18         wind = item['wind'].encode('utf8')
19
20         conn = MySQLdb.connect(
21                 host='localhost',
22                 port=3306,
23                 user='crawlUSER',
24                 passwd='crawl123',
25                 db='scrapyDB',
26                 charset = 'utf8')
27         cur = conn.cursor()
28         cur.execute("INSERT INTO
weather(cityDate,week,img,temperature,weather,wind)  values(%s,%s,%s,%s,%s,%s)",
(cityDate,week,img,temperature,weather,wind))
29         cur.close()
30         conn.commit()
31         conn.close()
32
33         return item
```

第 1 行指定了爬取数据的字符编码，第 8~9 行导入了所需的模块。第 20~31 行使用 MySQLdb 模块将数据写入了 MySQL 数据库中。最后在 settings.py 中将 pipelines2mysql.py 加

入到数据处理数列中去。修改后的 settings.py 内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for weather project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'weather'

SPIDER_MODULES = ['weather.spiders']
NEWSPIDER_MODULE = 'weather.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'weather (+http://www.yourdomain.com)'

#### user add
ITEM_PIPELINES = {
'weather.pipelines.WeatherPipeline':1,
'weather.pipelines2json.WeatherPipeline':2,
'weather.pipelines2mysql.WeatherPipeline':3
}
```

实际上就是把 pipelines2mysql 加入到 settings.py 的 ITEM_PIPELINES 项的字典中去就可以了。

最后运行 scrapy 爬虫，查看 MySQL 中的结果，执行命令：

```
scrapy crawl wuHanSpider
mysql -u crawlUSER -p
use scrapyDB;
select * from weaterh;
```

执行结果如图 5-36 所示。

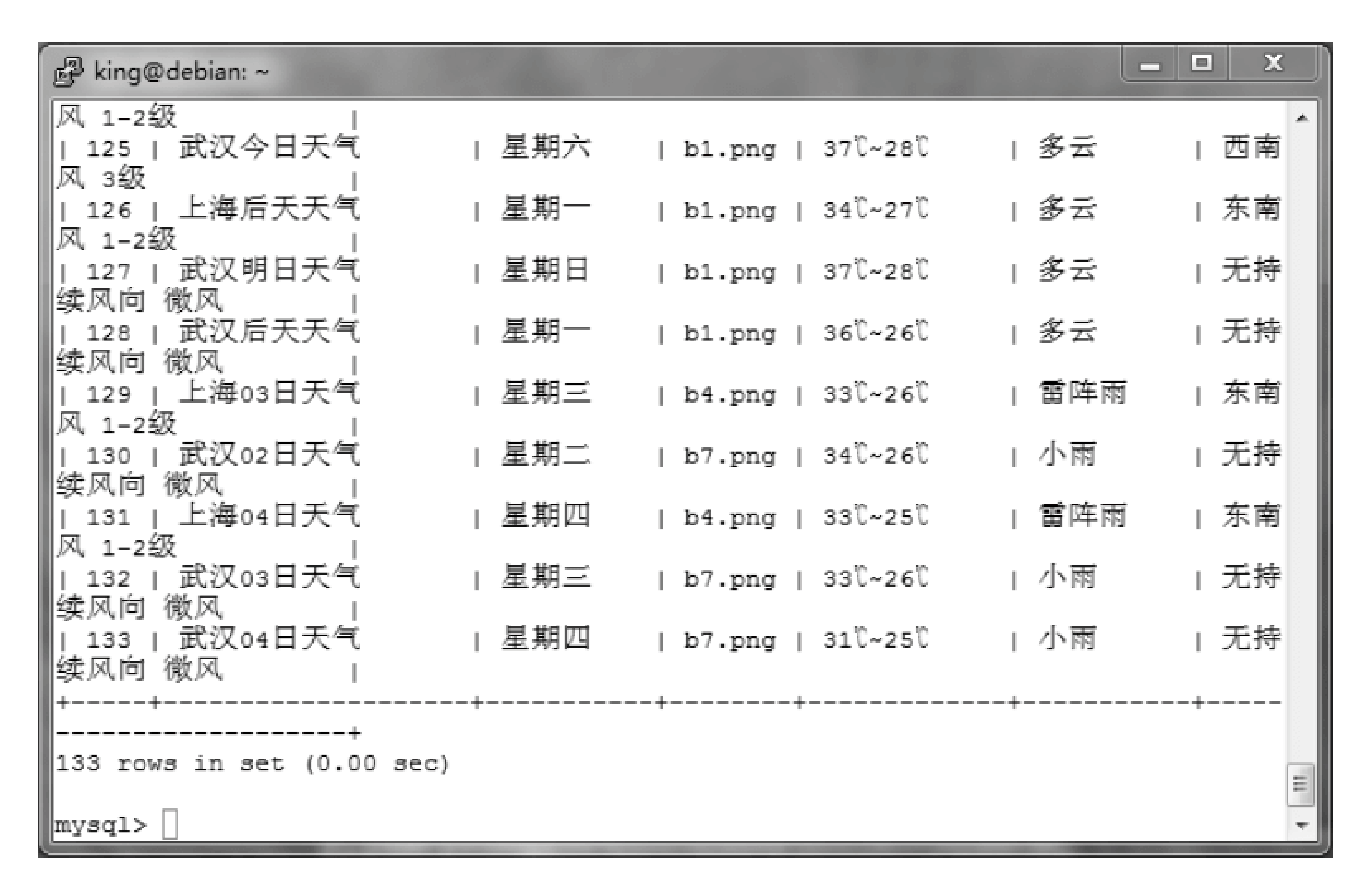


图 5-36　MySQL 中数据

MySQL 中显示 MySQLdb 模块存储数据有效。这个 Scrapy 项目到此就顺利完成了。

一般来说为了阅读方便，结果保存为 txt 就可以了。如果爬取的数据不多，需要存入表格备查，那可以保存为 cvs 或 json 比较方便。如果需要爬取的数据非常大，那还是老老实实考虑用 MySQL 吧。专业的软件做专业的事情。

5.5　Scrapy 爬虫实战三：获取代理

上节中的爬虫虽然将数据保存起来，但还是略有瑕疵。例如 cityDate 中显示了城市，显示了日期，但并没有显示具体的年月日。如果数据比较少，还可以慢慢地反推计算；如果数据多了，那就太麻烦了。这就涉及了爬取数据后的处理，本节我们讲解一下爬取数据后如何处理数据。

5.5.1　项目准备

本节将从网站上获取免费的代理服务器。使用 Scrapy 获取代理服务器后，一一验证哪些代理服务器可用，最终将可用的代理服务器保存到文件。

首先要做的是找到免费代理服务器的来源。浏览器中打开百度，搜索“免费代理服务器”，搜索结果如图 5-37 所示。

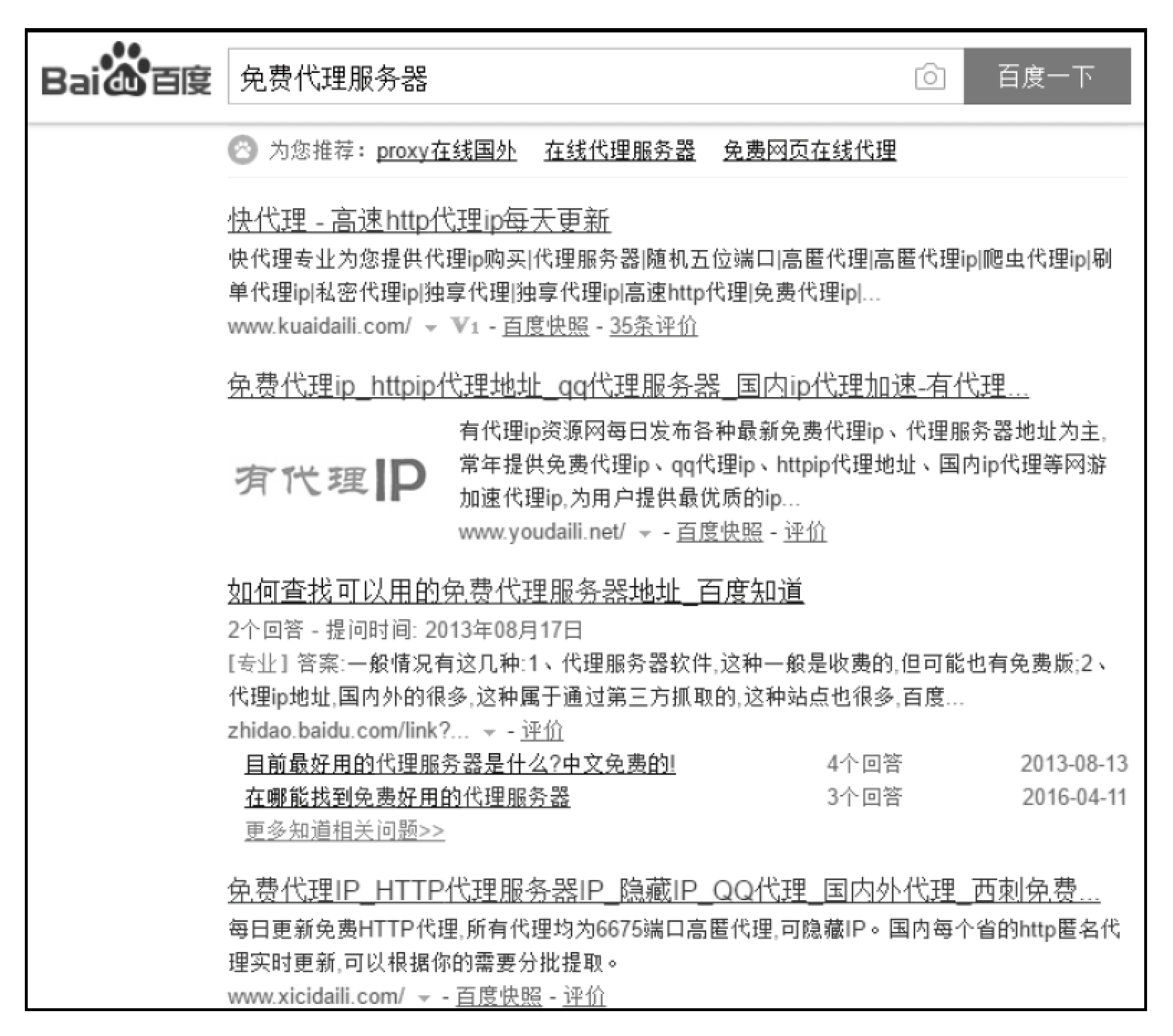


图 5-37　搜索免费代理服务器

先在 www.proxy360.cn 中获取代理服务器。如果数量不够，可以在 www.xicidaili.com 中获取代理服务器。

在浏览器中打开这两个站点，观察所需的项目。发现大部分的项目都相同，共有的项目有服务器 IP、服务器端口、是否匿名、服务器位置。xicidaili 站点中独有的项有服务协议。最后还应该添加获取服务器的来源站点。知道了这些内容，items.py 文件基本已经出来了。

5.5.2　创建编辑 Scrapy 爬虫

打开 Putty 连接到 Linux。在工作目录下创建 Scrapy 项目，并根据提示依照 spider 基础模版创建一个 spider。执行命令：

```
cd
cd code/crawler/scrapyProject
scrapy startproject getProxy
cd getProxy
scrapy genspider proxy360Spider proxy360.cn
```

这里创建了一个名为 getProxy 的 Scrapy 项目，并创建了一个名为 proxy360Spider.py 的 Spider 文件。

1．修改 items.py

根据前面的分析，items.py 应该包含 6 项。items.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class GetproxyItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
     ip = scrapy.Field()
     port = scrapy.Field()
     type = scrapy.Field()
     loction = scrapy.Field()
     protocol = scrapy.Field()
     source = scrapy.Field()
```

需要几项，就填入几项。最简模式就是只要代理 IP 和端口。这个文件没什么好解释的，比较简单直白。

2．修改 Spider 文件 proxy360Spider.py

先把 proxy360Spider.py 文件放到一边。使用 scrapy shell 命令查看一下连接网站返回的结果和数据，进入 getProxy 项目下的任意目录下，执行命令：

```
scrapy shell http://www.proxy360.cn/Region/China
```

执行结果如图 5-38 所示。

```
king@debian: ~/code/crawler/scrapyProject/getProxy/getProxy
tMiddleware, CookiesMiddleware, ChunkedTransferMiddleware, DownloaderStats
2016-07-31 19:22:32+0800 [scrapy] INFO: Enabled spider middlewares: HttpErrorMid
dleware, OffsiteMiddleware, RefererMiddleware, UrlLengthMiddleware, DepthMiddlew
are
2016-07-31 19:22:32+0800 [scrapy] INFO: Enabled item pipelines: GetproxyPipeline
2016-07-31 19:22:32+0800 [scrapy] DEBUG: Telnet console listening on 127.0.0.1:6
023
2016-07-31 19:22:32+0800 [scrapy] DEBUG: Web service listening on 127.0.0.1:6080
2016-07-31 19:22:32+0800 [default] INFO: Spider opened
2016-07-31 19:22:32+0800 [default] DEBUG: Crawled (200) <GET http://www.proxy360
.cn/Region/China> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x7fcce19f3910>
[s]   item       {}
[s]   request    <GET http://www.proxy360.cn/Region/China>
[s]   response   <200 http://www.proxy360.cn/Region/China>
[s]   settings   <scrapy.settings.Settings object at 0x7fcce23ec750>
[s]   spider     <Spider 'default' at 0x7fcce0b8d150>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

In [1]:
```

图 5-38　scrapy shell

从 response 的返回代码可以看出 request 请求正常返回。再查看一下 response 的数据内容，执行命令：

```
response.xapth('/*').extract()
```

执行结果如图 5-39 所示。

```
king@debian: ~/code/crawler/scrapyProject/getProxy/getProxy
n style="width:60px;" class="tbBottomLine">\r\n          \u603b\u7684\
u8bc4\u5206\r\n          </span>\r\n          <span style="width:30p
x;" class="tbBottomLine">\r\n          \u53ef\u7528\r\n
 </span>\r\n          <span style="width:100px; text-align:center; " class
="tbBottomLine">\r\n          \u901f\u5ea6\u6d4b\u8bc4\r\n
   </span>\r\n        </div>\r\n\r\n\r\n    \r\n        <div class="proxyli
stitem" name="list_proxy_ip">\r\n          <div style="float:left; display:blo
ck; width:630px;">\r\n          <span class="tbBottomLine" style="width:140px;
">\r\n          58.222.254.11\r\n          </span>\r\n          <span
class="tbBottomLine" style="width:50px;">\r\n          3128\r\n
   </span>\r\n          <span class="tbBottomLine " style="width:70px;">\r\n
       \u9ad8\u533f\r\n          </span>\r\n          <span class="
tbBottomLine " style="width:70px;">\r\n          \u4e2d\u56fd\r\n
  </span>\r\n          <span class="tbBottomLine " style="width:80px;">\r\n
       09\u670805\u65e5\r\n          </span>\r\n          <span class
="tbBottomLine " style="width:80px;">\r\n          3.44(70\u7968)\r\n
   </span>\r\n          <span class="tbBottomLine " style="width:60px;">\r
\n          3.44\r\n          </span>\r\n          <span class="tbBott
omLine " style="width:30px;">\r\n          14\u5929\r\n          </span>
\r\n          </div>\r\n          <div style="width:100px; float:left;">\r\n
       <div id="ctl00_ContentPlaceHolder1_repProxyList_ctl01_RatingSpee
d" title="3.44285714285714">\r\n\t\t<input type="hidden" name="ctl00$ContentPlac
eHolder1$repProxyList$ctl01$RatingSpeed_RatingExtender_ClientState" id="ctl00_Co
ntentPlaceHolder1_repProxyList_ctl01_RatingSpeed_RatingExtender_ClientState" val
```

图 5-39　response 数据

返回的数据中含有代理服务器（难道返回代码为 200 时，还有返回数据不含代理服务器的吗？这个还真有）。测试一下如何使用选择器在 response 中的得到所需的数据。在浏览器中打开 http://www.proxy360.cn/Region/China，在网页的任意空白处单击右键，选择“查看框架源代码”，打开页面的源代码网页，如图 5-40 所示。

```
view-source:www.proxy360.cn/Region/China
应用  从 Firefox 导入  百度  伪谷歌搜索  有道翻译  有道词典  study

        <div class="proxylistitem" name="list_proxy_ip">
            <div style="float:left; display:block; width:630px;">
            <span class="tbBottomLine" style="width:140px;">
                61.185.219.126
            </span>
            <span class="tbBottomLine" style="width:50px;">
                    3128
            </span>
            <span class="tbBottomLine " style="width:70px;">
                高匿
            </span>
            <span class="tbBottomLine " style="width:70px;">
                中国
            </span>
            <span class="tbBottomLine " style="width:80px;">
                09月05日
            </span>
            <span class="tbBottomLine " style="width:80px;">
                3.36(22票)
            </span>
            <span class="tbBottomLine " style="width:60px;">
                3.36
            </span>
            <span class="tbBottomLine " style="width:30px;">
                2天
            </span>
            </div>
            <div style="width:100px; float:left;">
                <div id="ctl00_ContentPlaceHolder1_repProxyList2_ctl01_RatingSpee
```

图 5-40　页面源代码

观察一下，似乎所有的数据块都是以<div class="proxylistitem" name="list_proxy_ip">这个 tag 开头的。在 scrapy shell 中测试一下，回到 scrapy shell 中，执行命令：

```
subSelector = response.xpath('//div[@class="proxylistitem" and @name
="list_proxy_ip"]')
subSelector.xpath('.//span[1]/text()').extract()[0]
subSelector.xpath('.//span[2]/text()').extract()[0]
subSelector.xpath('.//span[3]/text()').extract()[0]
subSelector.xpath('.//span[4]/text()').extract()[0]
```

执行结果如图 5-41 所示。

```
king@debian: ~/code/crawler/scrapyProject/getProxy/getProxy
In [9]: subSelector = response.xpath('//div[@class="proxylistitem" and @name="li
st_proxy_ip"]')

In [10]: subSelector.xpath('.//span[1]/text()').extract()[0]
Out[10]: u'\r\n[redacted]58.222.254.11\r\n[redacted]'

In [11]: subSelector.xpath('.//span[2]/text()').extract()[0]
Out[11]: u'\r\n                    3128\r\n              '

In [12]: subSelector.xpath('.//span[3]/text()').extract()[0]
Out[12]: u'\r\n                \u9ad8\u533f\r\n              '

In [13]: subSelector.xpath('.//span[4]/text()').extract()[0]
Out[13]: u'\r\n                \u4e2d\u56fd\r\n              '

In [14]:
```

图 5-41 proxy360Spider 测试选择器



所得数据左右两侧都有很多的空格。

现在如何用选择器从 response 中获取所需数据的方法也出来了，接下来可以开始编写 Spider 文件 proxy360Spider.py。proxy360Spider.py 的内容如下：

```
# -*- coding: utf-8 -*-
import scrapy
from getProxy.items import GetproxyItem

class Proxy360Spider(scrapy.Spider):
    name = "proxy360Spider"
    allowed_domains = ["proxy360.com"]
    nations =
['Brazil','China','America','Taiwan','Japan','Thailand','Vietnam','bahrein']
    start_urls = []
    for nation in nations:
        start_urls.append('http://www.proxy360.cn/Region/' + nation)


    def parse(self, response):
        subSelector = response.xpath('//div[@class="proxylistitem" and
@name="list_proxy_ip"]')
```

```
        items = []
        for sub in subSelector:
            item = GetproxyItem()
            item['ip'] = sub.xpath('.//span[1]/text()').extract()[0]
            item['port'] = sub.xpath('.//span[2]/text()').extract()[0]
            item['type'] = sub.xpath('.//span[3]/text()').extract()[0]
            item['loction'] = sub.xpath('.//span[4]/text()').extract()[0]
            item['protocol'] = 'HTTP'
            item['source'] = 'proxy360'
            items.append(item)
        return items
```

在 http://www.proxy360.cn/Region/China 页面中并没有显示服务器使用的协议。一般都是 HTTP 协议，所以 item['protocol']统一设置成了 HTTP。而数据来源都是 proxy360 网站，item['source']都设置成了 proxy360。

3．修改 pipelines.py，处理 Spider 的结果

这里还是将 Spider 的结果保存为 txt 格式，以便于阅读。pipelines.py 文件内容如下：

```
 1 # -*- coding: utf-8 -*-
 2
 3 # Define your item pipelines here
 4 #
 5 # Don't forget to add your pipeline to the ITEM_PIPELINES setting
 6 # See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html
 7
 8
 9 class GetproxyPipeline(object):
10     def process_item(self, item, spider):
11         fileName = 'proxy.txt'
12         with open(fileName, 'a') as fp:
13             fp.write(item['ip'].encode('utf8').strip() + '\t')
14             fp.write(item['port'].encode('utf8').strip() + '\t')
15             fp.write(item['protocol'].encode('utf8').strip() + '\t')
16             fp.write(item['type'].encode('utf8').strip() + '\t')
17             fp.write(item['loction'].encode('utf8').strip() + '\t')
18             fp.write(item['source'].encode('utf8').strip() + '\n')
19         return item
```

在 13~18 行中，写入文件时除了使用 encode('utf8')修改字符编码以外还使用了 strip()函数去除所得数据左右两边的空格。

4．修改 settings.py，决定由哪个文件来处理获取的数据

settings.py 稍作修改即可。将 pipelines.py 添加到 ITEM_PIPELINES 中去就能解决问题。settings.sp 文件内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for getProxy project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'getProxy'

SPIDER_MODULES = ['getProxy.spiders']
NEWSPIDER_MODULE = 'getProxy.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'getProxy (+http://www.yourdomain.com)'


### user add
ITEM_PIPELINES = {
'getProxy.pipelines.GetproxyPipeline':100
}
```

最后回到项目 getProxy 目录下，执行命令：

```
scrapy crawl proxy360Spider
ls
wc -l proxy.txt
more proxy.txt
```

执行结果如图 5-42 所示。



图 5-42　scrapy crawl proxy360Spider

得到的结果没什么问题。可花了这么长时间最后只得到了 144 个数据的结果，性价比也太低了点吧。不过没关系，再给它一个 Spider 就可以了，一个站点的数据不够就再加一个站点。如果还不够，那就继续增加站点吧。

5.5.3　多个 Spider

按照一个 Spider 的思路，得到的 proxy 数据不够多，则可以在 www.xicidaili.com 中获取代理补足。到项目 getProxy 目录下，执行命令：

```
cd
cd code/crawler/scrapyProject
cd getProxy
scrapy genspider xiciSpider xicidaili.com
```

创建一个名为 xiciSpider.py 的 Spider 文件。items.py 无须修改了，直接对 xiciSpider.py 做修改就可以了。我们还是先用 scrapy shell 命令来确定如何获取数据，执行命令：

```
scrapy shell http://www.xicidaili.com/nn/2
```

得到的结果如图 5-43 所示。

```
king@debian: ~/code/crawler/scrapyProject/getProxy
023
2016-07-31 20:28:10+0800 [scrapy] DEBUG: Web service listening on 127.0.0.1:6080
2016-07-31 20:28:10+0800 [xiciSpider] INFO: Spider opened
2016-07-31 20:28:10+0800 [xiciSpider] DEBUG: Retrying <GET http://www.xicidaili.
com/nn/2> (failed 1 times): 500 Internal Server Error
2016-07-31 20:28:10+0800 [xiciSpider] DEBUG: Retrying <GET http://www.xicidaili.
com/nn/2> (failed 2 times): 500 Internal Server Error
2016-07-31 20:28:10+0800 [xiciSpider] DEBUG: Gave up retrying <GET http://www.xi
cidaili.com/nn/2> (failed 3 times): 500 Internal Server Error
2016-07-31 20:28:10+0800 [xiciSpider] DEBUG: Crawled (500) <GET http://www.xicid
aili.com/nn/2> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x7f5b39055910>
[s]   item       {}
[s]   request    <GET http://www.xicidaili.com/nn/2>
[s]   response   <500 http://www.xicidaili.com/nn/2>
[s]   settings   <scrapy.settings.Settings object at 0x7f5b39a4e750>
[s]   spider     <XicispiderSpider 'xiciSpider' at 0x7f5b381ef190>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

In [1]:
```

图 5-43　scrapy shell

这里发现一个问题。respnose 返回的代码是 500，要知道返回码是 200 才是正常返回。在浏览器中打开 http://www.xicidaili.com/nn/2 是正常显示的，而该网页并不需要登录，那就是说并不涉及 cookie。用 scrapy shell 请求页面和用浏览器请求页面用的是同一 IP。也不存在 IP 封锁的问题。剩下的就只有 headers 中 User-Agent 的问题了。除去所有的不可能，最后那一选项大致就是正确答案。

在 Scrapy 中的确是有默认的 headers，但这个 headers 与浏览器的 headers 是有区别的。有的网站会检查 headers，如果是浏览器的 headers 网站则予以通过，而机器人或者说爬虫的

headers 则拒绝访问。所以在这里只需要给 scrapy 一个浏览器的 headers 就可以解决问题了。修改 settings.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for getProxy project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'getProxy'

SPIDER_MODULES = ['getProxy.spiders']
NEWSPIDER_MODULE = 'getProxy.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'getProxy (+http://www.yourdomain.com)'


### user add
USER_AGENT = 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)'
ITEM_PIPELINES = {
'getProxy.pipelines.GetproxyPipeline':100
}
```

只需要在 settings.py 里添加一个 USER_AGENT 项就可以了。如果可以，尽可能使用本机浏览器的 headers。这里使用的是随意选取的一个 headers，好在该网站没有根据不同的 headers 返回不同的内容。

好了，再回到 getProxy 项目的目录下，使用 scrapy shell 测试如何获取有效数据。执行命令：

```
scrapy shell http://www.xicidaili.com/nn/2
```

得到的结果如图 5-44 所示。

```
king@debian: ~/code/crawler/scrapyProject/getProxy
tMiddleware, CookiesMiddleware, ChunkedTransferMiddleware, DownloaderStats
2016-07-31 20:52:16+0800 [scrapy] INFO: Enabled spider middlewares: HttpErrorMid
dleware, OffsiteMiddleware, RefererMiddleware, UrlLengthMiddleware, DepthMiddlew
are
2016-07-31 20:52:16+0800 [scrapy] INFO: Enabled item pipelines: GetproxyPipeline
2016-07-31 20:52:16+0800 [scrapy] DEBUG: Telnet console listening on 127.0.0.1:6
023
2016-07-31 20:52:16+0800 [scrapy] DEBUG: Web service listening on 127.0.0.1:6080
2016-07-31 20:52:16+0800 [xiciSpider] INFO: Spider opened
2016-07-31 20:52:16+0800 [xiciSpider] DEBUG: Crawled (200) <GET http://www.xicid
aili.com/nn/2> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x7f212465a910>
[s]   item       {}
[s]   request    <GET http://www.xicidaili.com/nn/2>
[s]   response   <200 http://www.xicidaili.com/nn/2>
[s]   settings   <scrapy.settings.Settings object at 0x7f2125053750>
[s]   spider     <XicispiderSpider 'xiciSpider' at 0x7f21237f4190>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

In [1]:
```

图 5-44　修改 headers 后 scrapy shell

如图 5-44 所示，response 的返回码为 200，现在没问题了。浏览器中打开 http://www.xicidaili.com/nn/2，空白处单击右键，选择“查看页面源代码”，打开了页面的源代码网页，发现所需数据的块都是以<tr class="odd">或者<tr class="">开头的。在 scrapy shell 中执行命令：

```
subSelector = response.xpath('//tr[@class=""]|//tr[@class="odd"]')
subSelector[0].xpath('.//td[2]/text()').extract()[0]
subSelector[0].xpath('.//td[3]/text()').extract()[0]
subSelector[0].xpath('.//td[4]//text()').extract()[0]
subSelector[0].xpath('.//td[5]/text()').extract()[0]
subSelector[0].xpath('.//td[6]/text()').extract()[0]
```

执行结果如图 5-45 所示。

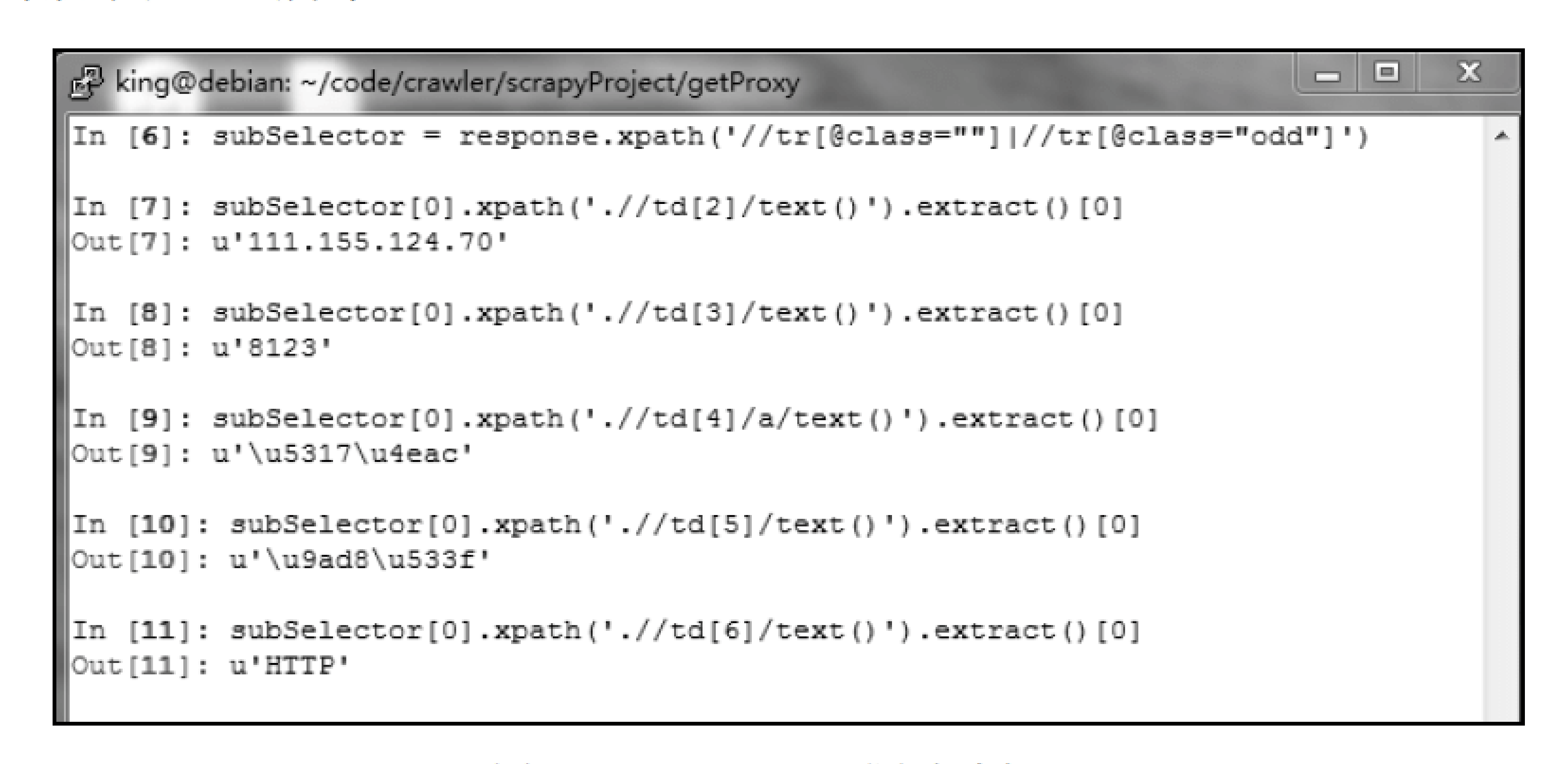


图 5-45　xiciSpider 测试选择器

现在 xiciSpider.py 怎么编写已经一目了然了。xiciSpider.py 的内容如下：

```
# -*- coding: utf-8 -*-
```

```
import scrapy
from getProxy.items import GetproxyItem


class XicispiderSpider(scrapy.Spider):
    name = "xiciSpider"
    allowed_domains = ["xicidaili.com"]
    wds = ['nn','nt','wn','wt']
    pages = 20
    start_urls = []
    for type in wds:
        for i in xrange(1,pages + 1):
            start_urls.append('http://www.xicidaili.com/' + type + '/' + str(i))


    def parse(self, response):
        subSelector = response.xpath('//tr[@class=""]|//tr[@class="odd"]')
        items = []
        for sub in subSelector:
            item = GetproxyItem()
            item['ip'] = sub.xpath('.//td[2]/text()').extract()[0]
            item['port'] = sub.xpath('.//td[3]/text()').extract()[0]
            item['type'] = sub.xpath('.//td[5]/text()').extract()[0]
            if sub.xpath('.//td[4]/a/text()'):
                item['loction'] = sub.xpath('//td[4]/a/text()').extract()[0]
            else:
                item['loction'] = sub.xpath('.//td[4]/text()').extract()[0]
            item['protocol'] = sub.xpath('.//td[6]/text()').extract()[0]
            item['source'] = 'xicidaili'
            items.append(item)
        return items
```

回到项目 getProxy 目录下，执行命令：

```
scrapy crawl xiciSpider
ls
wc -l proxy.txt
```

执行结果如图 5-46 所示。

```
king@debian: ~/code/crawler/scrapyProject/getProxy
2016-07-31 22:04:42+0800 [xiciSpider] INFO: Dumping Scrapy stats:
        {'downloader/request_bytes': 36972,
         'downloader/request_count': 80,
         'downloader/request_method_count/GET': 80,
         'downloader/response_bytes': 649788,
         'downloader/response_count': 80,
         'downloader/response_status_count/200': 80,
         'finish_reason': 'finished',
         'finish_time': datetime.datetime(2016, 7, 31, 14, 4, 42, 977442),
         'item_scraped_count': 8000,
         'log_count/DEBUG': 8082,
         'log_count/INFO': 8,
         'response_received_count': 80,
         'scheduler/dequeued': 80,
         'scheduler/dequeued/memory': 80,
         'scheduler/enqueued': 80,
         'scheduler/enqueued/memory': 80,
         'start_time': datetime.datetime(2016, 7, 31, 14, 3, 20, 377657)}
2016-07-31 22:04:42+0800 [xiciSpider] INFO: Spider closed (finished)
king@debian:~/code/crawler/scrapyProject/getProxy$ ls
getProxy  scrapy.cfg   proxy.txt
king@debian:~/code/crawler/scrapyProject/getProxy$ wc -l proxy.txt
8144  proxy.txt
king@debian:~/code/crawler/scrapyProject/getProxy$
```

图 5-46　scrapy crawl xiciSpider



xicidaili.com 同一 IP，短时间内频繁爬取，超过一定次数就会被封锁 IP。万一被封锁了，那就重启路由器或者光猫换个 IP 吧。

从文件保存的记录数字来看，应该是没问题的。但这么多的记录，或者说这么多的代理服务器有多少是可用的呢？这就是下一小节的问题了。

5.5.4　处理 Spider 数据

如何来验证上一小节中已经获取到的代理服务器地址，最简单的方法当然是在 pipelines 文件里直接修改。但不幸的是验证一个代理服务器是否有效所需的时间和将一行记录写入文件的时间相差得太远了。前者所需的时间是以秒计算的，后者是以微秒计算。这样一来还不如先将所有的代理服务器保存到文件，然后另外写一个 Python 程序来验证代理。

进入 getProxy 项目的目录下，创建 Python 验证程序。执行命令：

```
cd
cd code/crawler/scrapyProject
cd getProxy
vi testProxy.py
```

代理服务器验证程序 testProxy.py 的内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


import urllib2
import re
```

```
import threading


class TestProxy(object):
    def __init__(self):
        self.sFile = r'proxy.txt'
        self.dFile = r'alive.txt'
        self.URL = r'http://www.baidu.com/'
        self.threads = 10
        self.timeout = 3
        self.regex = re.compile(r'baidu.com')
        self.aliveList = []

        self.run()

    def run(self):
        with open(self.sFile, 'r') as fp:
            lines = fp.readlines()
            line = lines.pop()
            while lines:
                for i in xrange(self.threads):
                    t = threading.Thread(target=self.linkWithProxy, args=(line,))
                    t.start()
                    if lines:
                        line = lines.pop()
                    else:
                        continue

        with open(self.dFile, 'w') as fp:
            for i in xrange(len(self.aliveList)):
                fp.write(self.aliveList[i])



    def linkWithProxy(self, line):
        lineList = line.split('\t')
        protocol = lineList[2].lower()
        server = protocol + r'://' + lineList[0] + ':' + lineList[1]
        opener = urllib2.build_opener(urllib2.ProxyHandler({protocol:server}))
        urllib2.install_opener(opener)
        try:
```

```
            response = urllib2.urlopen(self.URL, timeout=self.timeout)
        except:
            print('%s connect failed' %server)
            return
        else:
            try:
                str = response.read()
            except:
                print('%s connect failed' %server)
                return
            if self.regex.search(str):
                print('%s connect success ..........' %server)
                self.aliveList.append(line)
if __name__ == '__main__':
    TP = TestProxy()
```

执行命令：

```
python testProxy.py
```

利用多线程来验证来源文件 proxy.txt 里的代理。经过验证，有 181 个代理可以使用。将 self.threads 设置为 10，使用 10 个进程并发，大概需要 1 分钟左右，速度还可以接受。这个速度已经很快了，没有必要将 self.threads 设置得太大，以免占用太多的系统资源。最终得到文件 alive.txt。这个程序还比较简陋，有很大的改进空间。例如，同一网站下爬取的代理服务器也许不会有重复的情况，但多个网站爬取的代理服务器就有可能重复。这种情况可以在程序内加上一个去重复的函数。

5.6 Scrapy 爬虫实战四：糗事百科

上节中得到了一个经过验证的 proxy 文件。本节将使用得到的代理来爬取网站内容，目标站点就定为一个笑话网站糗事百科。

5.6.1 目标分析

糗事百科这个站点类似于上节的“西刺代理”。它必须要指定一个浏览器的 headers 才能返回正确的数据。另外上节中的 getProxy 项目中已经获取了一些可使用的代理服务器，浪费是可耻的。本节将使用代理来爬取糗事百科中的笑话。

在浏览器中打开糗事百科网站，如图 5-47 所示。



图 5-47 数据来源站点

从图 5-47 中可以看出目标数据来源的网址为 http://www.qiushibaike.com/hot/page/3/?s=4900120，这里的?s=4900120 应该只是从 Cookies 里提取的用户标识。去除这个尾巴，用浏览器打开 http://www.qiushibaike.com/hot/page/3/。页面完全一样，没有任何影响。

可以获取的项有发布者名字、笑话内容、笑话图片（如果有图片就下载）、单击好笑的次数、谈论的次数。items.py 就是这些了。

5.6.2 创建编辑 Scrapy 爬虫

进入 Scrapy 的工作目录，创建项目名为 qiushi 的 Scrapy 项目，通过 Spider 模版创建 qiushiSpider.py 文件。执行命令：

```
cd
cd code/crawler/scrapyProject
scrapy startproject qiushi
cd qiushi
scrapy genspider qiushiSpider qiushibaike.com
```

首先要编辑的还是 items.py，items.py 文件的内容如下：

```
#-*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy
```

```


class QiushiItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
     author = scrapy.Field()
     content = scrapy.Field()
     img = scrapy.Field()
     funNum = scrapy.Field()
     talkNum = scrapy.Field()
```

不管后面怎么变化，需要添加什么功能，在 items.py 这个文件上是没有任何区别的。在上节的 getProxy 项目中，为了获取“西刺代理”站点上的代理服务器，在 settings.py 中添加了 USER_AGENT 项，给 Scrapy 添加了一个浏览器的 headers。本节的 Scrapy 项目不仅需要添加浏览器的 headers，还要使用 proxy，这就涉及了 Scrapy 中间件。

Scrapy 项目本身有很多的中间件。这些中间件设置了很多的环境。一般最常见的中间件就是下载器中间件。本节项目所需添加 headers，使用 proxy 都是在这个中间件中修改。

5.6.3　Scrapy 项目中间件——添加 headers

在 Scrapy 项目中，掌管 proxy 的中间件是 scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware。直接修改这个中间件，不是不可以，不过为了一个项目就去修改整个环境变量，也太小题大做了。我们完全可以自己写个中间件，让它运行，然后将 Scrapy 默认的中间件关闭掉就可以了。

先进入 qiushi 项目下的 settgings.py 同层目录，创建文件夹 middlewares，在 middlewares 目录下创建__init__.py 和 customMiddlewares.py 文件，其中__init__.py 的作用是将整个 middlewares 目录当成一个模块使用。customMiddlewares.py 就是自定义的中间件。customMiddlewares.py 的内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


from scrapy.contrib.downloadermiddleware.useragent import UserAgentMiddleware

class CustomUserAgent(UserAgentMiddleware):
    def process_request(self,request,spider):
        ua = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3"
        request.headers.setdefault('User-Agent', ua)
```

修改 settings.py，将系统默认的中间件 scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware 关闭，用自己创建的中间件 qiushi.middlewares.customMiddlewares.CustomUserAgent 代替。Settings.py 内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for qiushi project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'qiushi'

SPIDER_MODULES = ['qiushi.spiders']
NEWSPIDER_MODULE = 'qiushi.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'qiushi (+http://www.yourdomain.com)'



### user define
DOWNLOADER_MIDDLEWARES = {
    'qiushi.middlewares.customMiddlewares.CustomUserAgent': 3,
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,
}
```

因为改用了自定义的中间件取代 Scrapy 的中间件，所以需要将 Scrapy 的中间件改为 None，将其关闭。

编辑 pipelines.py 文件，pipelines.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

```

```
import time
import urllib2
import os

class QiushiPipeline(object):
    def process_item(self, item, spider):
        today = time.strftime('%Y%m%d', time.localtime())
        fileName = today + 'qiubai.txt'
        imgDir = 'IMG'
        if os.path.isdir(imgDir):
            pass
        else:
            os.mkdir(imgDir)
        with open(fileName, 'a') as fp:
            fp.write('-'*50 + '\n' + '*'*50 + '\n')
            fp.write("author:\t %s\n" %(item['author'].encode('utf8')))
            fp.write("content:\t %s\n" %(item['content'].encode('utf8')))
            try:
                imgUrl = item['img'][1]
            except IndexError:
                pass
            else:
                imgName = os.path.basename(imgUrl)
                fp.write("img:\t %s\n" %(imgName))
                imgPathName = imgDir + os.sep + imgName
                with open(imgPathName, 'wb') as fpi:
                    response = urllib2.urlopen(imgUrl)
                    fpi.write(response.read())
            fp.write("fun:%s\t talk:%s\n" %(item['funNum'],item['talkNum']))
            fp.write('*'*50 + '\n' + '-'*50 + '\n'*10)
        return item
```

最后将 QiushiPipeline 添加到 settings.py 中去，修改后的 settings.py 内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for qiushi project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
```

```
#

BOT_NAME = 'qiushi'

SPIDER_MODULES = ['qiushi.spiders']
NEWSPIDER_MODULE = 'qiushi.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'qiushi (+http://www.yourdomain.com)'



### user define
DOWNLOADER_MIDDLEWARES = {
    'qiushi.middlewares.customMiddlewares.CustomUserAgent': 3,
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,
}

ITEM_PIPELINES = {
'qiushi.pipelines.QiushiPipeline':10,
}
```

所有文件准备完毕了，回到 qiushi 项目下，执行命令：

```
scrapy crawl qiushiSpider
ls
```

结果如图 5-48 所示。

```
king@debian: ~/code/crawler/scrapyProject/qiushi
        {'downloader/request_bytes': 30250,
         'downloader/request_count': 65,
         'downloader/request_method_count/GET': 65,
         'downloader/response_bytes': 342865,
         'downloader/response_count': 65,
         'downloader/response_status_count/200': 22,
         'downloader/response_status_count/301': 1,
         'downloader/response_status_count/503': 42,
         'finish_reason': 'finished',
         'finish_time': datetime.datetime(2016, 8, 1, 9, 6, 20, 316252),
         'item_scraped_count': 363,
         'log_count/DEBUG': 446,
         'log_count/ERROR': 77,
         'log_count/INFO': 7,
         'response_received_count': 30,
         'scheduler/dequeued': 65,
         'scheduler/dequeued/memory': 65,
         'scheduler/enqueued': 65,
         'scheduler/enqueued/memory': 65,
         'start_time': datetime.datetime(2016, 8, 1, 9, 5, 59, 392965)}
2016-08-01 17:06:20+0800 [qiushiSpider] INFO: Spider closed (finished)
king@debian:~/code/crawler/scrapyProject/qiushi$ ls
20160801qiubai.txt  IMG  qiushi  scrapy.cfg
king@debian:~/code/crawler/scrapyProject/qiushi$
```

图 5-48　数据保存结果

从最后运行结果看来，已达到预期目标，获取了数据。自定义的中间件成功运行。

5.6.4　Scrapy 项目中间件——添加 proxy

上一小节中使用自定义的中间件给 Scrapy 添加了浏览器的 headers。本小节将使用自定义的中间件给 Scrapy 添加一个 proxy。

Scrapy 默认环境下，proxy 的设置是由中间件 scrapy.contrib.downloadermiddleware.httpproxy.HttpProxyMiddleware 控制的。参照上一小节的方法，还是自定义一个中间件。因为只使用单独一个代理，就不再添加新的文件了。直接在 middlewares.customMiddlewares 中添加一个类就可以了。

既然是使用 proxy，那得先找到一个可使用 proxy，在上个 Scrapy 项目 getProxy 中已经找到很多了，我们在 getProxy 项目的最终文档 usefulProxy.txt 中随意挑选一个即可。例如：114.33.202.73:8118。

修改自定义的中间件文档 customMiddlewares.py。修改完毕后的 customMiddlewares.py 内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


from scrapy.contrib.downloadermiddleware.useragent import UserAgentMiddleware

class CustomUserAgent(UserAgentMiddleware):
    def process_request(self,request,spider):
        ua = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3"
        request.headers.setdefault('User-Agent', ua)

class CustomProxy(object):
    def process_request(self,request,spider):
        request.meta['proxy'] = 'http://114.33.202.73:8118'
```

接下来再修改 settings.py 文件，将新添加的中间件 CustomProxy 添加到 DOWNLOADER_MIDDLEWARES 中去。这里与之前的 CustomUserAgent 不同的是，CustomUserAgent 需要禁止系统的 UserAgentMiddleware，而 CustomProxy 则需要在系统的 HttpProxyMiddleware 之前执行。修改完毕的 settings.py 内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for qiushi project
```

```
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'qiushi'

SPIDER_MODULES = ['qiushi.spiders']
NEWSPIDER_MODULE = 'qiushi.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'qiushi (+http://www.yourdomain.com)'



### user define
DOWNLOADER_MIDDLEWARES = {
    'qiushi.middlewares.customMiddlewares.CustomProxy': 10,
    'qiushi.middlewares.customMiddlewares.CustomUserAgent': 30,
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,
    'scrapy.contrib.downloadermiddleware.httpproxy.HttpProxyMiddleware': 20
}

ITEM_PIPELINES = {
'qiushi.pipelines.QiushiPipeline':10,
}
```

最后回到 Scrapy 项目 qiushi 的目录下，执行命令：

```
Scrapy crawl qiushiSpider
```

执行的结果如图 5-49 所示。

```
king@debian: ~/code/crawler/scrapyProject/qiushi
king@debian:~/code/crawler/scrapyProject/qiushi$ scrapy crawl qiushiSpider
2016-08-01 22:57:01+0800 [scrapy] INFO: Scrapy 0.24.2 started (bot: qiushi)
2016-08-01 22:57:01+0800 [scrapy] INFO: Optional features available: ssl, http11
, boto, django
2016-08-01 22:57:01+0800 [scrapy] INFO: Overridden settings: {'NEWSPIDER_MODULE'
: 'qiushi.spiders', 'SPIDER_MODULES': ['qiushi.spiders'], 'BOT_NAME': 'qiushi'}
2016-08-01 22:57:01+0800 [scrapy] INFO: Enabled extensions: LogStats, TelnetCons
ole, CloseSpider, WebService, CoreStats, SpiderState
2016-08-01 22:57:02+0800 [scrapy] INFO: Enabled downloader middlewares: CustomPr
oxy, CustomUserAgent, HttpAuthMiddleware, DownloadTimeoutMiddleware, RetryMiddle
ware, DefaultHeadersMiddleware, MetaRefreshMiddleware, HttpCompressionMiddleware
, RedirectMiddleware, CookiesMiddleware, ChunkedTransferMiddleware, DownloaderSt
ats
2016-08-01 22:57:02+0800 [scrapy] INFO: Enabled spider middlewares: HttpErrorMid
dleware, OffsiteMiddleware, RefererMiddleware, UrlLengthMiddleware, DepthMiddlew
are
2016-08-01 22:57:02+0800 [scrapy] INFO: Enabled item pipelines: QiushiPipeline
2016-08-01 22:57:02+0800 [qiushiSpider] INFO: Spider opened
2016-08-01 22:57:02+0800 [qiushiSpider] INFO: Crawled 0 pages (at 0 pages/min),
scraped 0 items (at 0 items/min)
2016-08-01 22:57:02+0800 [scrapy] DEBUG: Telnet console listening on 127.0.0.1:6
023
2016-08-01 22:57:02+0800 [scrapy] DEBUG: Web service listening on 127.0.0.1:6080
^C2016-08-01 22:57:04+0800 [scrapy] INFO: Received SIGINT, shutting down gracefu
```

图 5-49　运行中间件

从图 5-49 中可以看出自定义的两个中间件都已经运行了。

5.7 scrapy 爬虫实战五：爬虫攻防

对于一般用户而言，网络爬虫是个好工具，它可以方便地从网站上获取自己想要的信息。可对于网站而言，网络爬虫占用了太多的资源，也没可能从这些爬虫获取点击量，增加广告收入。据有关调查研究证明，网络上超过 60%以上的访问量都是爬虫造成的，也难怪网站方对网络爬虫恨之入骨，“杀”之而后快了。

网站方采取种种措施拒绝网络爬虫的访问，而网络高手们则毫不示弱，改进网络爬虫，赋予它更强的功能、更快的速度，以及更隐蔽的手段。在这场爬虫与反爬虫的战争中，双方的比分交替领先，最终谁会赢得胜利，大家将拭目以待。

5.7.1 创建一般爬虫

我们先写一个小爬虫程序，假设网站方的各种限制，然后再来看看如何破解网站方的限制，让大家自由地使用爬虫工具。网站限制的爬虫肯定不包括我们这种只有几次访问的爬虫。一般来说，小于 100 次访问的爬虫都无须为此担心，这个的爬虫纯粹是做演示。

以爬取美剧天堂站点为例，使用 Scrapy 爬虫来爬取最近更新的美剧，来源网页是 http://www.meijutt.com/new100.html。进入 Scrapy 工作目录，创建 meiju100 项目。执行命令：

```
cd
cd code/crawler/scrapyProject
scrapy startproject meiju100
cd meiju100
```

```
scrapy genspider meiju100Spider meijutt.com
tree ./meiju100
```

执行的结果如图 5-50 所示。

```
king@debian: ~/code/crawler/scrapyProject/meiju100

You can start your first spider with:
    cd meiju100
    scrapy genspider example example.com
king@debian:~/code/crawler/scrapyProject$ cd meiju100
king@debian:~/code/crawler/scrapyProject/meiju100$ scrapy genspider meiju100Spid
er meijutt.com
Created spider 'meiju100Spider' using template 'basic' in module:
  meiju100.spiders.meiju100Spider
king@debian:~/code/crawler/scrapyProject/meiju100$ tree ./meiju100/
./meiju100/
├── __init__.py
├── __init__.pyc
├── items.py
├── pipelines.py
├── settings.py
├── settings.pyc
└── spiders
    ├── __init__.py
    ├── __init__.pyc
    └── meiju100Spider.py

1 directory, 9 files
king@debian:~/code/crawler/scrapyProject/meiju100$
```

图 5-50　tree meiju100 项目

修改后的 items.py 的内容如下：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class Meiju100Item(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
     storyName = scrapy.Field()
     storyState = scrapy.Field()
     tvStation = scrapy.Field()
     updateTime = scrapy.Field()
```

修改后的 meiju100Spider.py 内容如下：

```
# -*- coding: utf-8 -*-
import scrapy

```

```
from meiju100.items import Meiju100Item

class Meiju100spiderSpider(scrapy.Spider):
    name = "meiju100Spider"
    allowed_domains = ["meijutt.com"]
    start_urls = (
        'http://www.meijutt.com/new100.html',
    )

    def parse(self, response):
        subSelector = response.xpath('//li/div[@class="lasted-num fn-left"]')
        items = []
        for sub in subSelector:
            item = Meiju100Item()
            item['storyName'] = sub.xpath('../h5/a/text()').extract()[0]
            item['storyState'] = sub.xpath('../span[@class="state1 new100state1"]/text()').extract()[0]
            item['tvStation'] = sub.xpath('../span[@class="mjtv"]/a/text()').extract()
            item['updateTime'] = sub.xpath('../div[@class="lasted-time new100time fn-right"]/text()').extract()[0]
            items.append(item)
        return items
```

修改后的 pipelines.py 内容如下：

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

import time

class Meiju100Pipeline(object):
    def process_item(self, item, spider):
        today = time.strftime('%Y%m%d', time.localtime())
        fileName = today + 'meiju.txt'
        with open(fileName, 'a') as fp:
            fp.write("%s \t" %(item['storyName'].encode('utf8')))
            fp.write("%s \t" %(item['storyState'].encode('utf8')))
            if len(item['tvStation']) == 0:
```

```
                fp.write("unknow \t")
            else:
                fp.write("%s \t" %(item['tvStation'][0]).encode('utf8'))
            fp.write("%s \n" %(item['updateTime'].encode('utf8')))
        return item
```

修改后的 settings.py 内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for meiju100 project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'meiju100'

SPIDER_MODULES = ['meiju100.spiders']
NEWSPIDER_MODULE = 'meiju100.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'meiju100 (+http://www.yourdomain.com)'


### user define
ITEM_PIPELINES = {
'meiju100.pipelines.Meiju100Pipeline':10
}
```

这个美剧爬虫已经修改完毕了。回到 meiju 项目的主目录下执行命令：

```
scrapy crawl meijuSpider
cat *.txt
```

执行结果如图 5-51 所示。

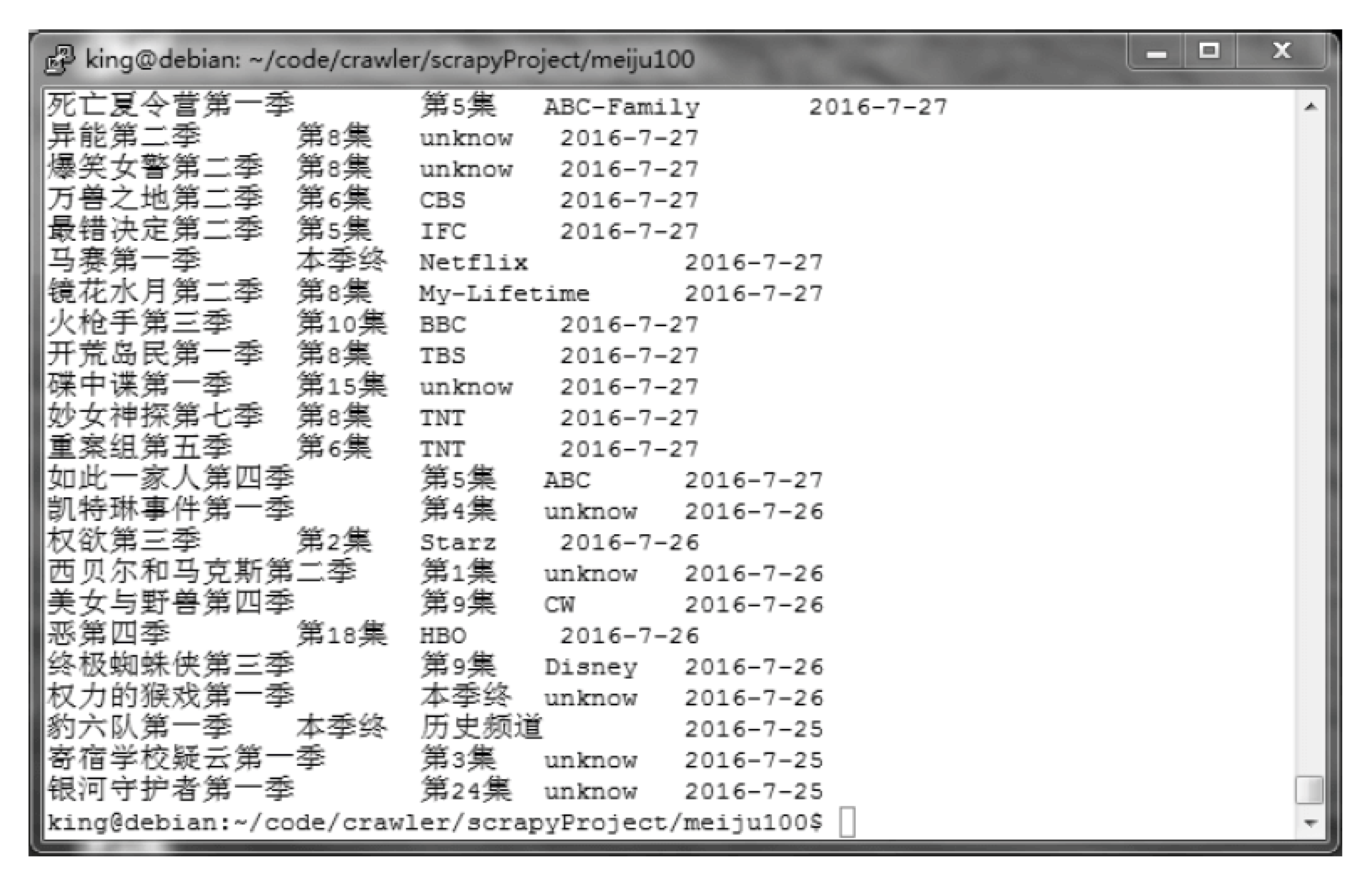


图 5-51 meiju 项目结果

项目运行成功。下面来测试反爬虫和反反爬虫技术。

5.7.2 封锁间隔时间破解

Scrapy 在两次请求之间的时间设置是 DOWNLOAD_DELAY。如果不考虑反爬虫的因素，这个值当然是越小越好。如果把 DOWNLOAD_DELAY 设置成了 0.1，也就是每 0.1 秒向网站请求一次网页。网站管理员只要不瞎，稍微过滤一下日志，就会为用户如此侮辱他的智商而愤恨不已。

如果对爬虫的结果需求不是那么急，也希望“打枪的不要，悄悄地进村”，那还是把这一项的值设置得稍微大一点吧。在 settings.py 的尾部追加这一项即可。

```
DOWNLOAD_DELAY = 5
```

5.7.3 封锁 Cookies 破解

众所周知，网站是通过 Cookies 来确定用户身份的。Scrapy 爬虫在爬取数据时使用同一个 Cookies 发送请求。这种做法和把 DOWNLOAD_DELAY 设置成 0.1 没什么区别。

不过要破解这种原理的反爬虫也很简单，直接禁用 Cookies 就可以了。在 settings.py 的尾部追加一项即可。

```
COOKIES_ENABLED = False
```

5.7.4 封锁 user-agent 破解

user-agent 是浏览器的身份标识。网站就是通过 user-agent 来确定浏览器类型的。有很多的网站都会拒绝不符合一定标准的 user-agent 请求网页。在前面的 Scrapy 项目中曾冒充浏览器访问网站。但如果网站将频繁访问网站的 user-agent 作为爬虫的标志，然后将其拉入黑名

单又该怎么办呢？

这个也很简单。可以准备一大堆的 user-agent，然后随机挑选一个使用，使用一次就更换，这样不就解决了。

首先还是在 meiju 项目下 settings.py 的同级目录创建 middlewares 目录，进入 middlewares 目录，创建__init__.py，将 middlewares 目录变成一个 Python 模块。创建资源文件 resource.py 和中间件文件 customUserAgent.py。

将多个浏览器的 user-agent 放入资源文件 resource.py 中加入列表待用。修改后的 resource.py 文件内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

UserAgents = [
 "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; AcooBrowser; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",
 "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506)",
 "Mozilla/4.0 (compatible; MSIE 7.0; AOL 9.5; AOLBuild 4337.35; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",
 "Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)",
 "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 2.0.50727; Media Center PC 6.0)",
 "Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 1.0.3705; .NET CLR 1.1.4322)",
 "Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 5.2; .NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.2; .NET CLR 3.0.04506.30)",
 "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN) AppleWebKit/523.15 (KHTML, like Gecko, Safari/419.3) Arora/0.3 (Change: 287 c9dfb30)",
 "Mozilla/5.0 (X11; U; Linux; en-US) AppleWebKit/527+ (KHTML, like Gecko, Safari/419.3) Arora/0.6",
 "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2pre) Gecko/20070215 K-Ninja/2.1.1",
 "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9) Gecko/20080705 Firefox/3.0 Kapiko/3.0",
 "Mozilla/5.0 (X11; Linux i686; U;) Gecko/20070322 Kazehakase/0.4.5",
 "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.8) Gecko Fedora/1.9.0.8-1.fc10 Kazehakase/0.5.6",
 "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11",
```

```
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/535.20 (KHTML, like Gecko) Chrome/19.0.1036.7 Safari/535.20",
"Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; fr) Presto/2.9.168 Version/11.52",
]
```

修改 customUserAgent.py，将资源文件中的 user-agent 随机选择一个出来，作为 Scrapy 的 user-agent。修改后的 customUserAgent.py 内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

from scrapy.contrib.downloadermiddleware.useragent import UserAgentMiddleware
from meiju100.middlewares.resource import UserAgents
import random

class RandomUserAgent(UserAgentMiddleware):
    def process_request(self,request,spider):
        ua = random.choice(UserAgents)
        request.headers.setdefault('User-Agent', ua)
```

最后修改 settings.py 文件，将 RandomUserAgent 加入 DOWNLOADER_MIDDLEWARES 项中。修改后的 settings.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for meiju100 project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'meiju100'

SPIDER_MODULES = ['meiju100.spiders']
NEWSPIDER_MODULE = 'meiju100.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'meiju100 (+http://www.yourdomain.com)'
```

```



### user define
ITEM_PIPELINES = {
'meiju100.pipelines.Meiju100Pipeline':10
}

DOWNLOADER_MIDDLEWARES = {
'meiju100.middlewares.customUserAgent.RandomUserAgent':30,
'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,
}
```

修改到这一步，使用 Scrapy 会随机调用一个 user-agent，稍微小心一点，Scrapy 再也不会因为 user-agent 而被网站拒绝服务了。

5.7.5 封锁 IP 破解

在反爬虫中，最容易被发觉的实际上是 IP。同一 IP 短时间内访问同一站点，如果数目少，管理员可能会以为是网吧或者大型的局域网在访问而放你一马。数目多了，那肯定是爬虫了。个人用户可以用重启猫的方法换 IP，专线用户总不能让 ISP 给换专线吧，因此最方便的方法就是使用代理了。

之前的项目中曾使用过代理爬取网站，本节将准备一个代理池，从中随机地选取一个代理使用。爬取一次，选取一个不同的代理。进入之前创建的 middlewares 目录中，在资源文件 resource.py 中加入一个 IP 池，也就是一个代理服务器的列表。在前面的项目中已经获取很多免费的代理服务器了，请随意取用，不用客气。

修改后的 resource.py 的内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

UserAgents = [
 "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; AcooBrowser; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",
 "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506)",
 "Mozilla/4.0 (compatible; MSIE 7.0; AOL 9.5; AOLBuild 4337.35; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",
 "Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)",
 "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 2.0.50727; Media Center PC 6.0)",
```

```
"Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 1.0.3705; .NET CLR 1.1.4322)",
"Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 5.2; .NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.2; .NET CLR 3.0.04506.30)",
"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN) AppleWebKit/523.15 (KHTML, like Gecko, Safari/419.3) Arora/0.3 (Change: 287 c9dfb30)",
"Mozilla/5.0 (X11; U; Linux; en-US) AppleWebKit/527+ (KHTML, like Gecko, Safari/419.3) Arora/0.6",
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2pre) Gecko/20070215 K-Ninja/2.1.1",
"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9) Gecko/20080705 Firefox/3.0 Kapiko/3.0",
"Mozilla/5.0 (X11; Linux i686; U;) Gecko/20070322 Kazehakase/0.4.5",
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.8) Gecko Fedora/1.9.0.8-1.fc10 Kazehakase/0.5.6",
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11",
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/535.20 (KHTML, like Gecko) Chrome/19.0.1036.7 Safari/535.20",
"Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; fr) Presto/2.9.168 Version/11.52",
]

PROXIES = [
'110.73.7.47:8123',
'110.73.42.145:8123',
'182.89.3.95:8123',
'39.1.40.234:8080',
'39.1.37.165:8080'
] #这里还可以添加更多的代理
```

创建一个中间件 customProxy.py。这个中间件的作用就是让 Scrapy 爬取网站时随机使用 IP 池中的代理。修改后的 customProxy.py 的内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


from meiju100.middlewares.resource import PROXIES
import random

class RandomProxy(object):
```

```
    def process_request(self,request,spider):
        proxy = random.choice(PROXIES)
        request.meta['proxy'] = 'http://%s' %proxy
```

最后还是修改 settings.py 文件，将 customProxy 加入到 DOWNLOADER_MIDDLEWARES 项中。修改后的 settings.py 文件内容如下：

```
# -*- coding: utf-8 -*-

# Scrapy settings for meiju100 project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

BOT_NAME = 'meiju100'

SPIDER_MODULES = ['meiju100.spiders']
NEWSPIDER_MODULE = 'meiju100.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'meiju100 (+http://www.yourdomain.com)'


### user define
ITEM_PIPELINES = {
'meiju100.pipelines.Meiju100Pipeline':10
}

DOWNLOADER_MIDDLEWARES = {
'meiju100.middlewares.customProxy.RandomProxy':10,
'meiju100.middlewares.customUserAgent.RandomUserAgent':30,
'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,
'scrapy.contrib.downloadermiddleware.httpproxy.HttpProxyMiddleware': 20
}
```

修改完毕后，Scrapy 就可以随机地使用 IP 池中的代理了。执行命令 scrapy crawl meiju100Spider 获取最新的美剧。

回到项目 meiju100 的主目录下，使用 tree 命令查看 meiju100 项目的文件，如图 5-52 所示。

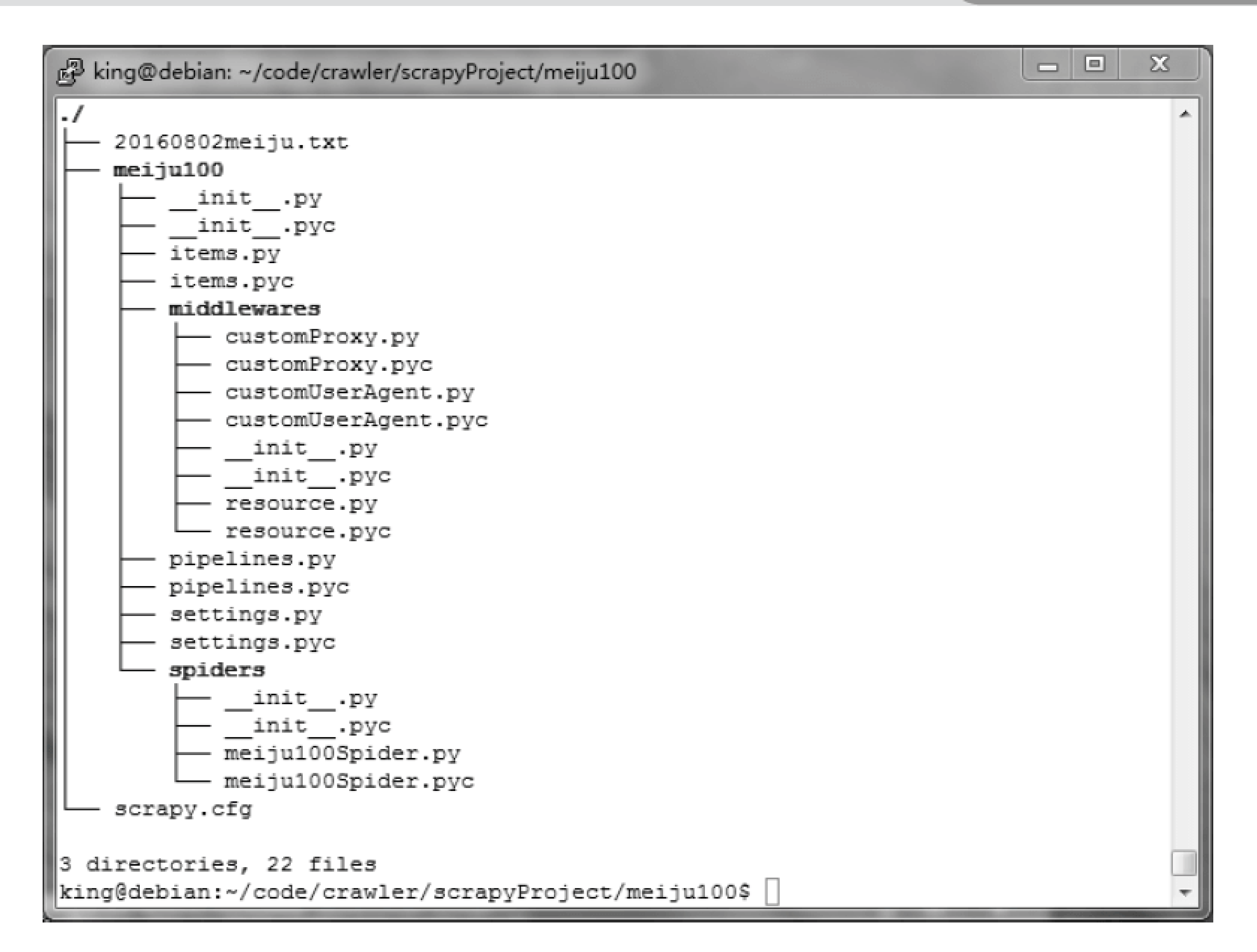


图 5-52　tree meiju100

实际反爬虫的方法远不止这一些，只不过个人用户掌握这些也够用了。个人用户动则成千上万的网页爬取毕竟是少数。

5.8 本章小结

本章详细介绍了 Scrapy 爬虫框架的使用，由易到难演示了 Scrapy 爬虫爬取网页的过程，并通过爬虫与反爬虫的攻守过程，让读者一窥 Scrapy 中间件的使用方法。从使用的难度来说，Scrapy 可以算得上最简单的爬虫了，简单到只需做填空题就能得到数据，而且对于特殊爬虫的特殊要求也能很好支持。

第 6 章

◄ Beautiful Soup爬虫 ►

上一章节讲解了 Python 的爬虫框架 Scrapy。本章将详细讲解另一个 Python 爬虫 Beautiful Soup。与 Scrapy 不同的是 Beautiful Soup 并不是一个框架，而是一个模块。因此，Beautiful Soup 不能再做填空题了，只能从头到尾的写作文了。

Beautiful Soup 最新版本是 4.4.0，一般被简称为 bs4。bs 仅支持 Python 2.7，如果想使用 Python 3.0 版本的 Beautiful Soup，那就只能使用 Beautiful Soup 3 了。Bs4 在网上的教程不多，好不容易找到几个，内容还都是重复的。这里主要是参考 bs4 的官网教程，官网网址为 http://beautifulsoup.readthedocs.io/zh_CN/latest/。实际上它也没什么很难的地方，与 Scrapy 相比，除了选择过滤有所不同外，就是一普通的 Python 程序。

6.1 安装 Beautiful Soup 环境

bs4 并不是软件，它只是一个第三方的模块。既然是模块，那安装起来就比较简单了。前面说的 pip、ease_install 都可以（推荐使用 pip）。

6.1.1 Windows 下安装 Beautiful Soup

在 Windows 下安装 Bearutiful Soup 最简单的方法还是使用 pip 安装。打开 cmd.exe，执行命令：

```
pip install beautifulsoup4
```

执行结果如图 6-1 所示。

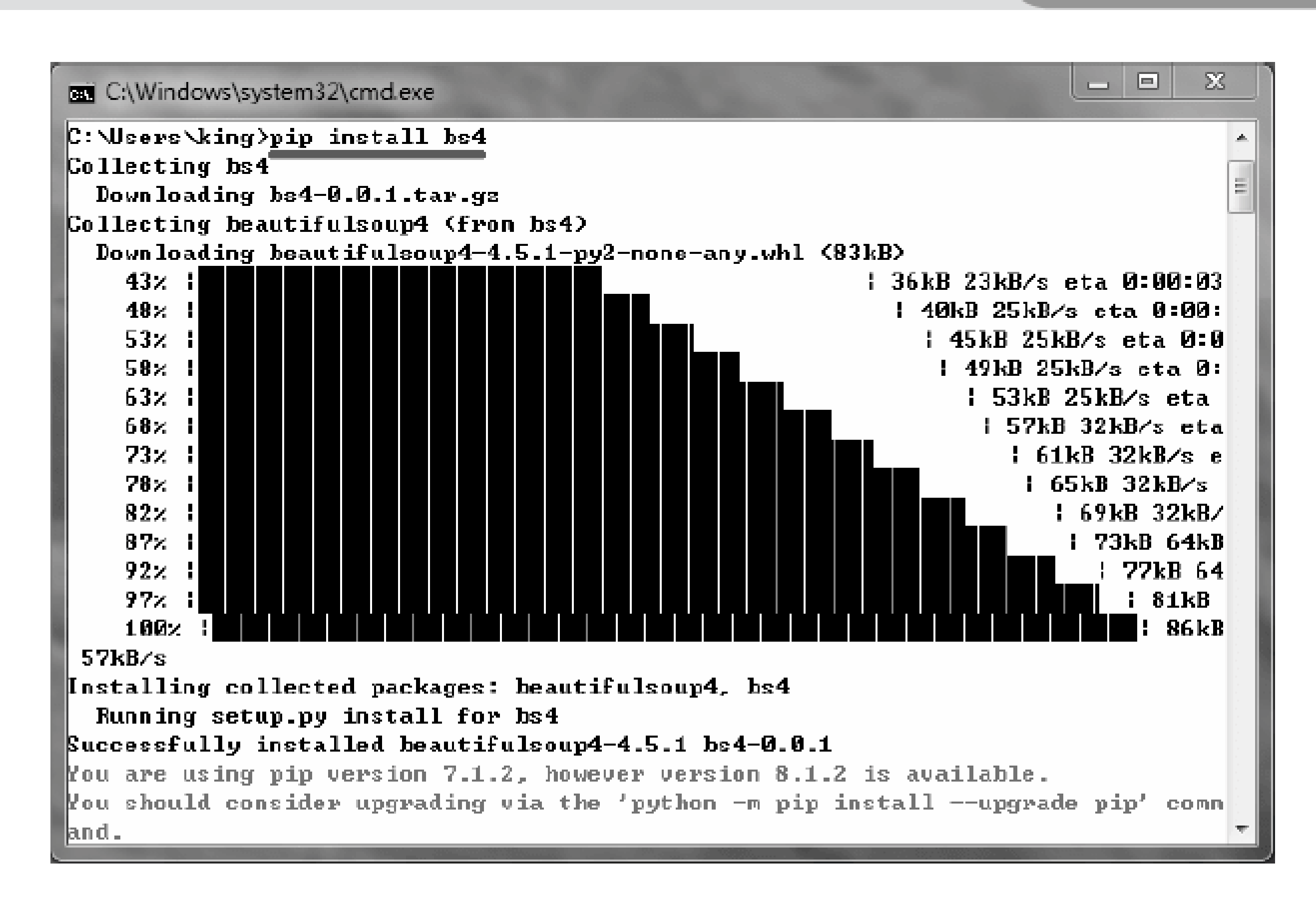


图 6-1　Windows 安装 bs4

bs4 已安装到 Windows 中，可以直接使用了。

6.1.2　Linux 下安装 Beautiful Soup

Linux 中安装还是借助于 Debian 的数据库，以便于管理。在终端中以 root 用户（如果普通用户有权限，也可以使用 sudo 命令安装）执行命令：

```
apt-get install python-bs4
```

执行结果如图 6-2 所示。

```
king@debian8: ~
root@debian8:/home/king# apt-get install python-bs4
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列【新】软件包将被安装：
  python-bs4
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 96 个软件包未被升级。
需要下载 0 B/77.7 kB 的软件包。
解压缩后会消耗掉 348 kB 的额外空间。
正在选中未选择的软件包 python-bs4。
(正在读取数据库 ... 系统当前共安装有 157310 个文件和目录。)
正准备解包 .../python-bs4_4.3.2-2_all.deb  ...
正在解包 python-bs4 (4.3.2-2) ...
正在设置 python-bs4 (4.3.2-2) ...
root@debian8:/home/king#
```

图 6-2　Linux 安装 bs4

基于 Debian 一贯的保守策略，apt-get 安装的并不是最新版本，而是目前最稳定的版本 4.3.2。

6.1.3　最强大的 IDE——Eclipse

Python 环境下有很多优秀的 IDE，如 Eclipse、Komodo、Sublime、Pycharm、vim、Emacs 等，其中 vim 和 Emacs 虽然是跨平台的，但配置复杂，而且界面也比较简陋，不符合美学原则。Pycharm、Komodo 在编译 Python 时还不错。但笔者更希望使用一款能包打天下的兼容所有语言的 IDE，而 Eclipse 不负众望，大而全配合插件后无所不包，无须安装到系统，可直接使用。最重要的是 Eclipse 免费啊，让人既没有使用盗版的负疚感，也能安然享受全部的服务。Eclipse 是跨平台的 IDE，能在所有系统下运行。本章中所有的程序如不特殊注

明都将在 Windows 下运行。

1．安装 Eclipse

打开 Eclipse 的官网下载页面 http://www.eclipse.org/downloads/，直接单击下载按钮，如图 6-3 所示。

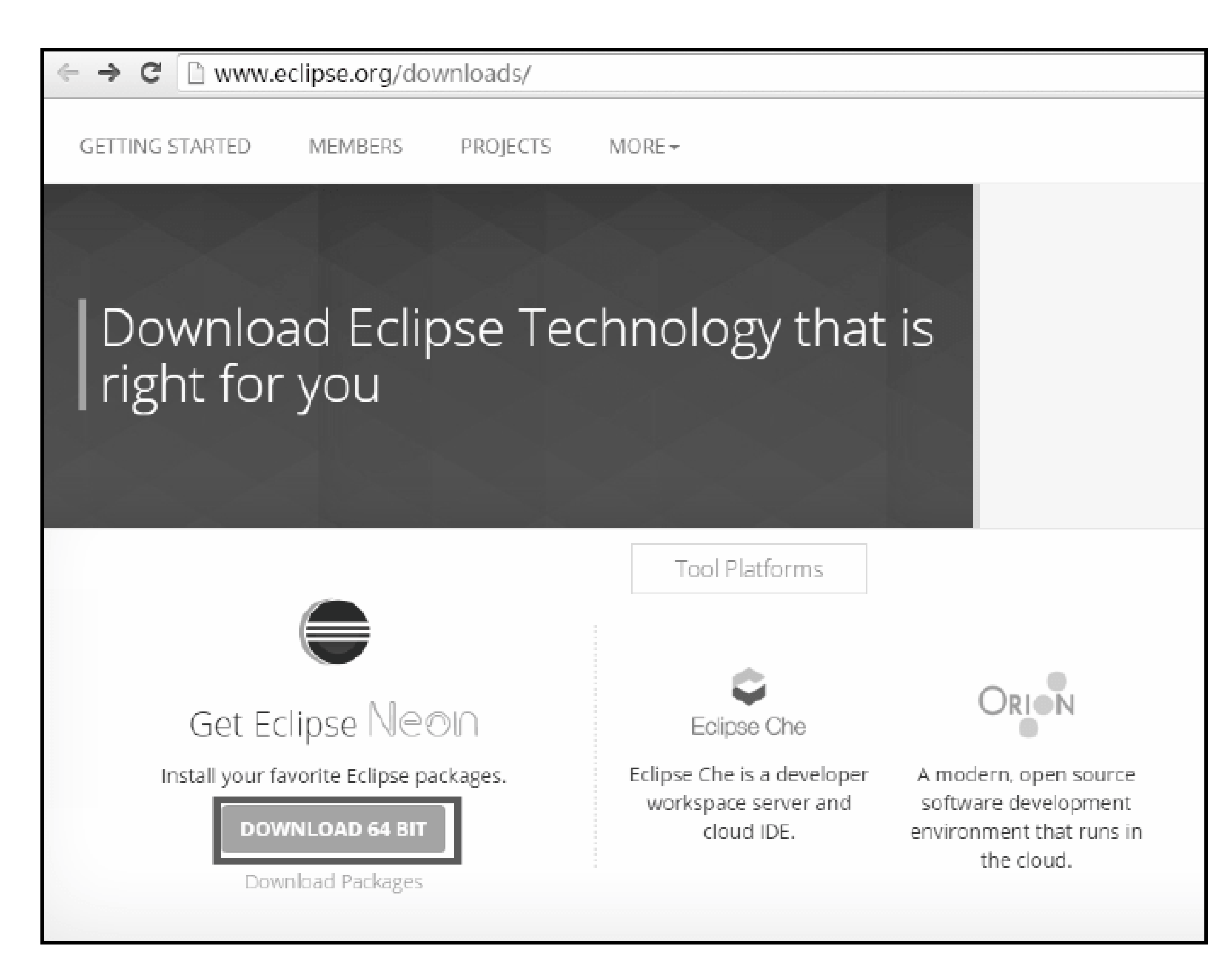


图 6-3 官网下载 Eclipse

网站会根据访问站点的系统（从访问者的 headers 就可以得出操作系统）推荐安装程序。本次下载网站推荐的是 eclipse-inst-win64.exe（前面说过 Eclipse 无须安装，是绿色程序并非笔误。这个所谓的安装程序基本就是个解压缩文件）。左键双击安装程序，要求选择 Eclipse 的版本，如图 6-4 所示。



图 6-4 选择 Eclipse 版本

单击 Eclipse IDE for java Developers 就可以了。因为是 for java，所以还得下载 java 依赖包。如果网速给力，用不了几分钟就可以下载完毕。下载完毕后，安装程序要求选择安装位置，如图 6-5 所示。

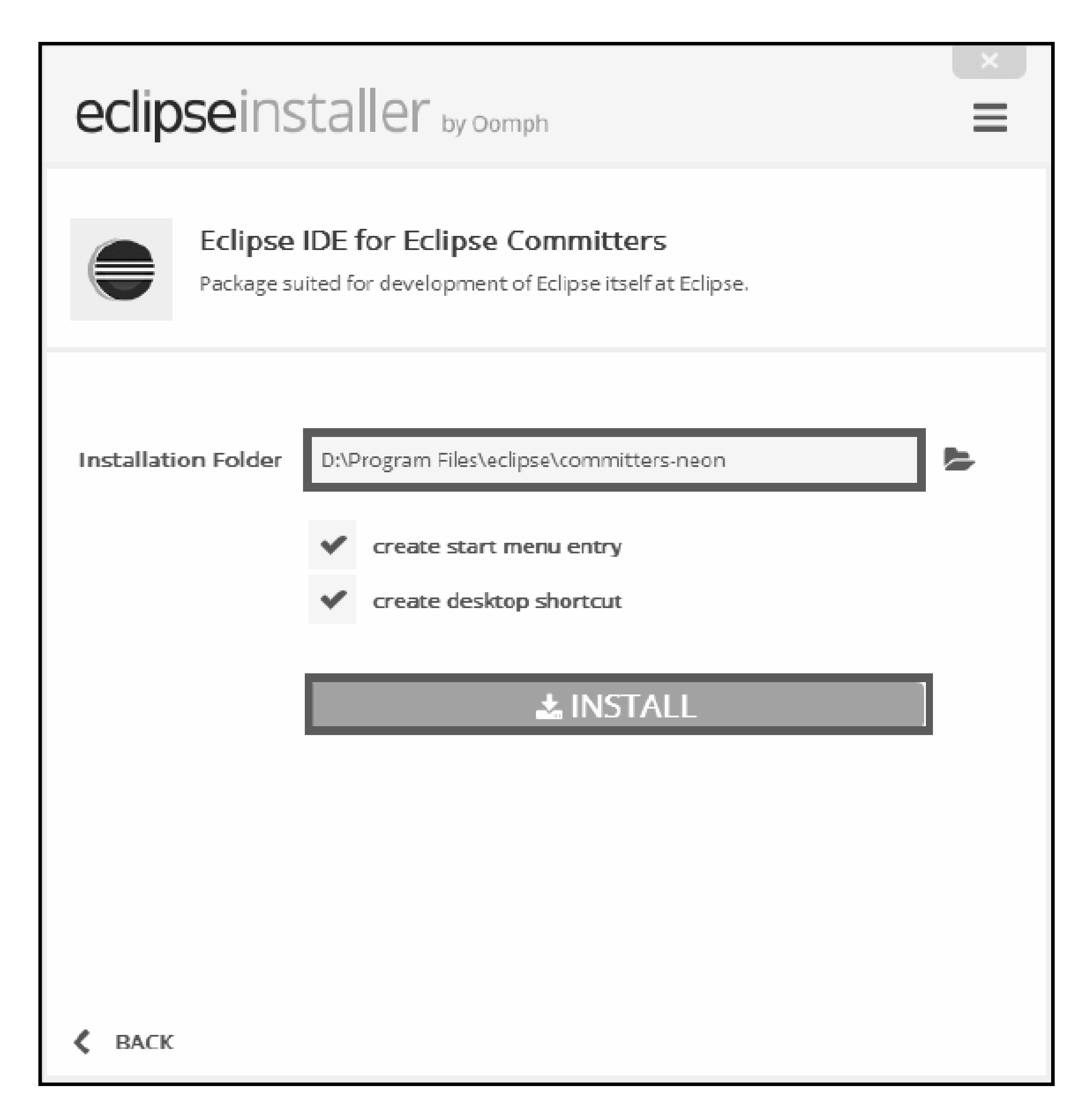


图 6-5　选择安装位置

填入合适的安装位置后，单击 INSTALL 按钮，稍待片刻 Eclipse 就安装完毕。双击桌面上的 Eclipse 图标运行 Eclipse。首次运行时会提示选择工作目录，如图 6-6 所示。

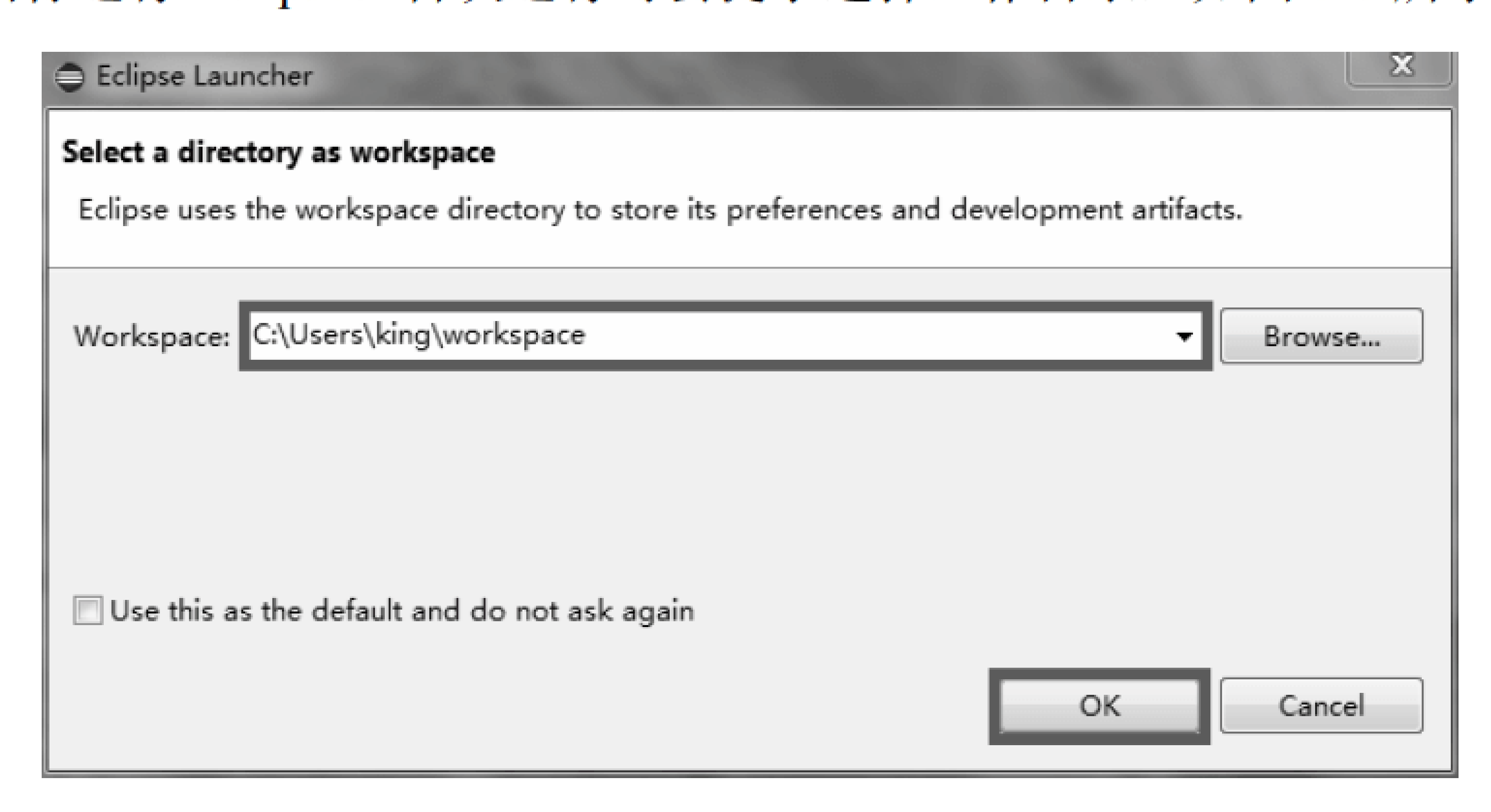


图 6-6　选择工作目录

填入 Eclipse 的工作目录，单击 OK 按钮，Eclipse 界面如图 6-7 所示。

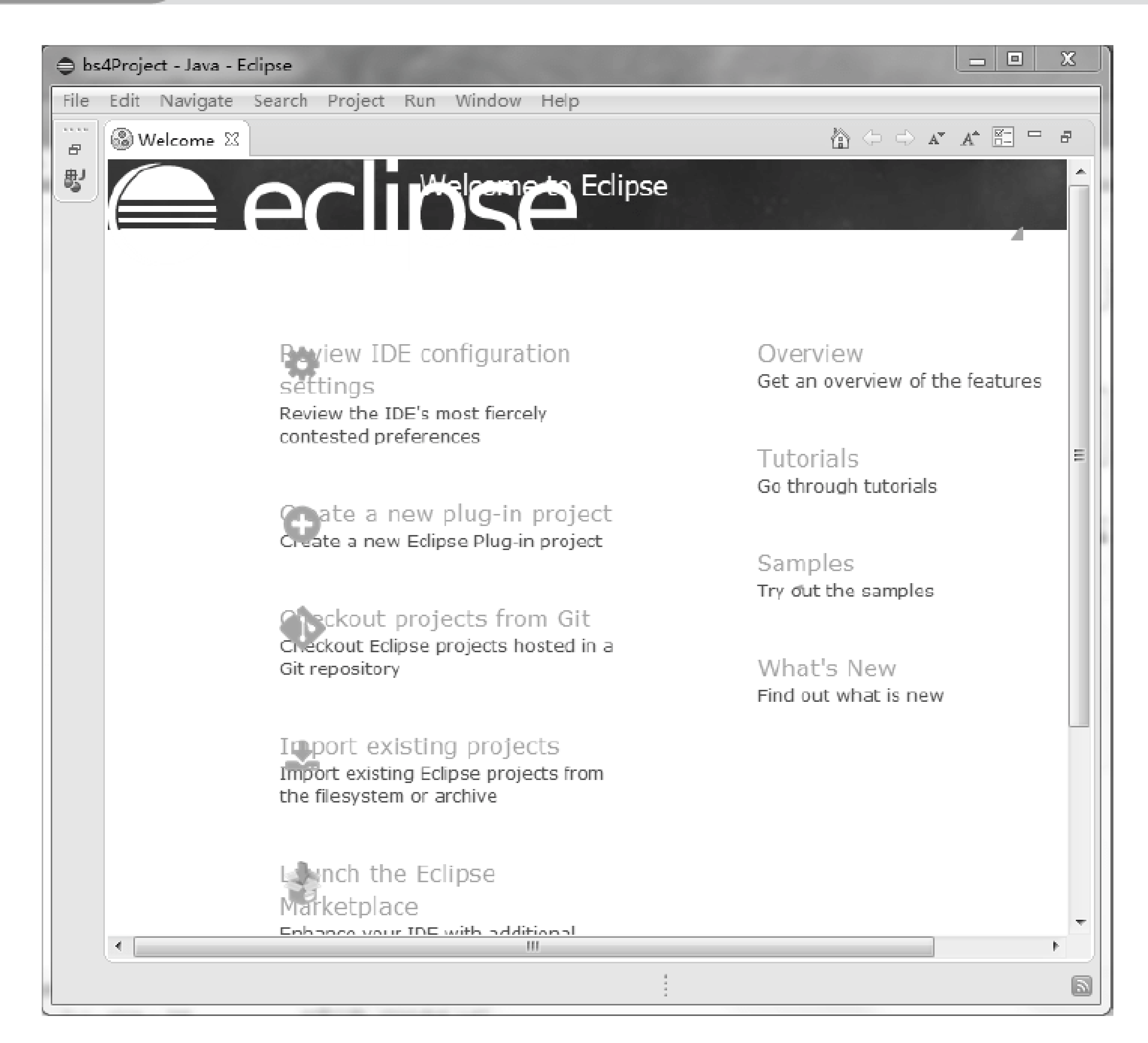


图 6-7　Eclipse 界面

Eclipse 安装完毕，下一步将安装 Eclipse 的 Python 插件 Pydev。

2．安装 Pydev 插件

在 Eclipse 的菜单上单击 Help|Install New Software 选项，如图 6-8 所示。

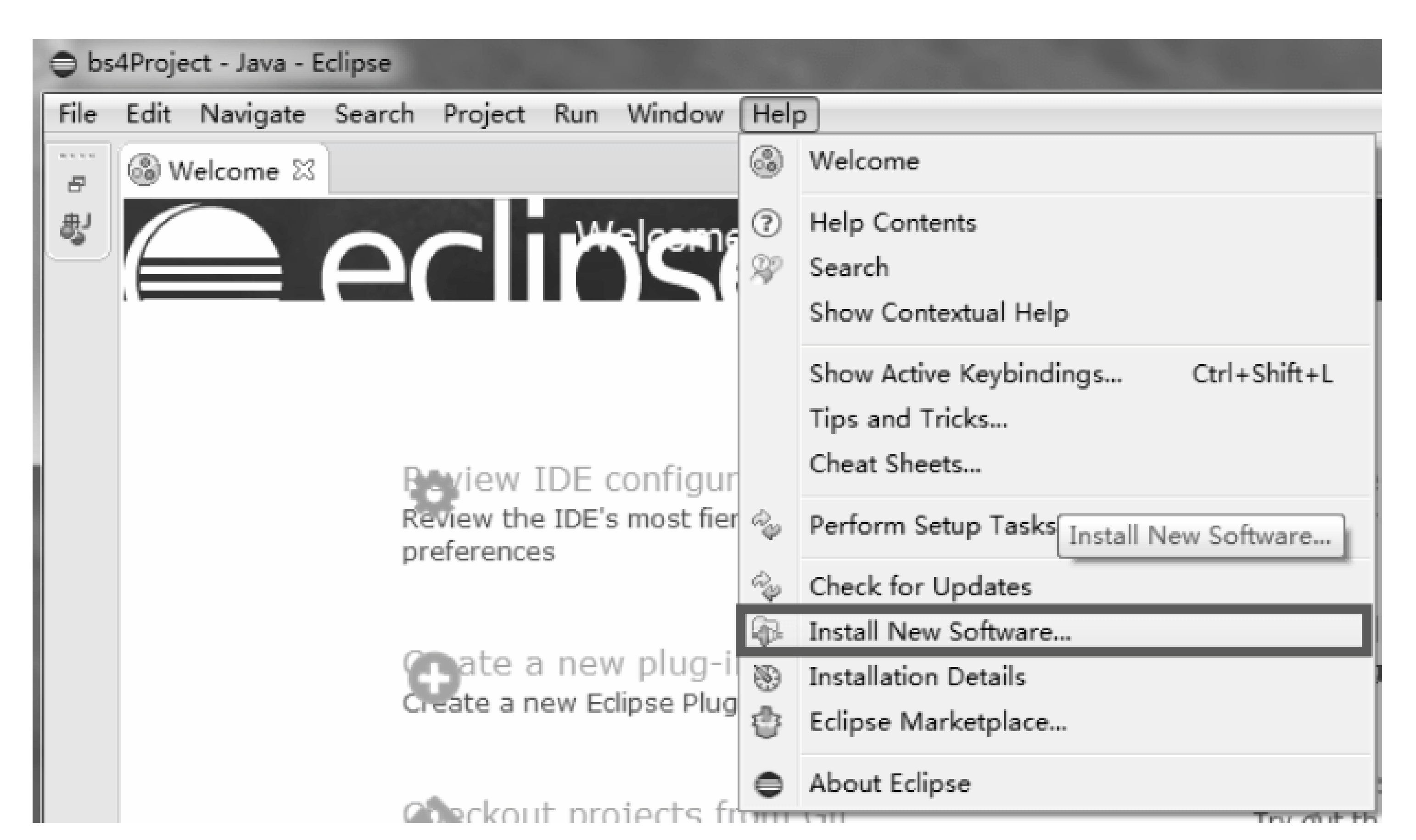


图 6-8　安装 Python 插件

打开了 Eclipse 的插件安装界面，单击 Add 按钮，加入 Eclipse 的 Python 插件 Pydev 的安装源，如图 6-9 所示。

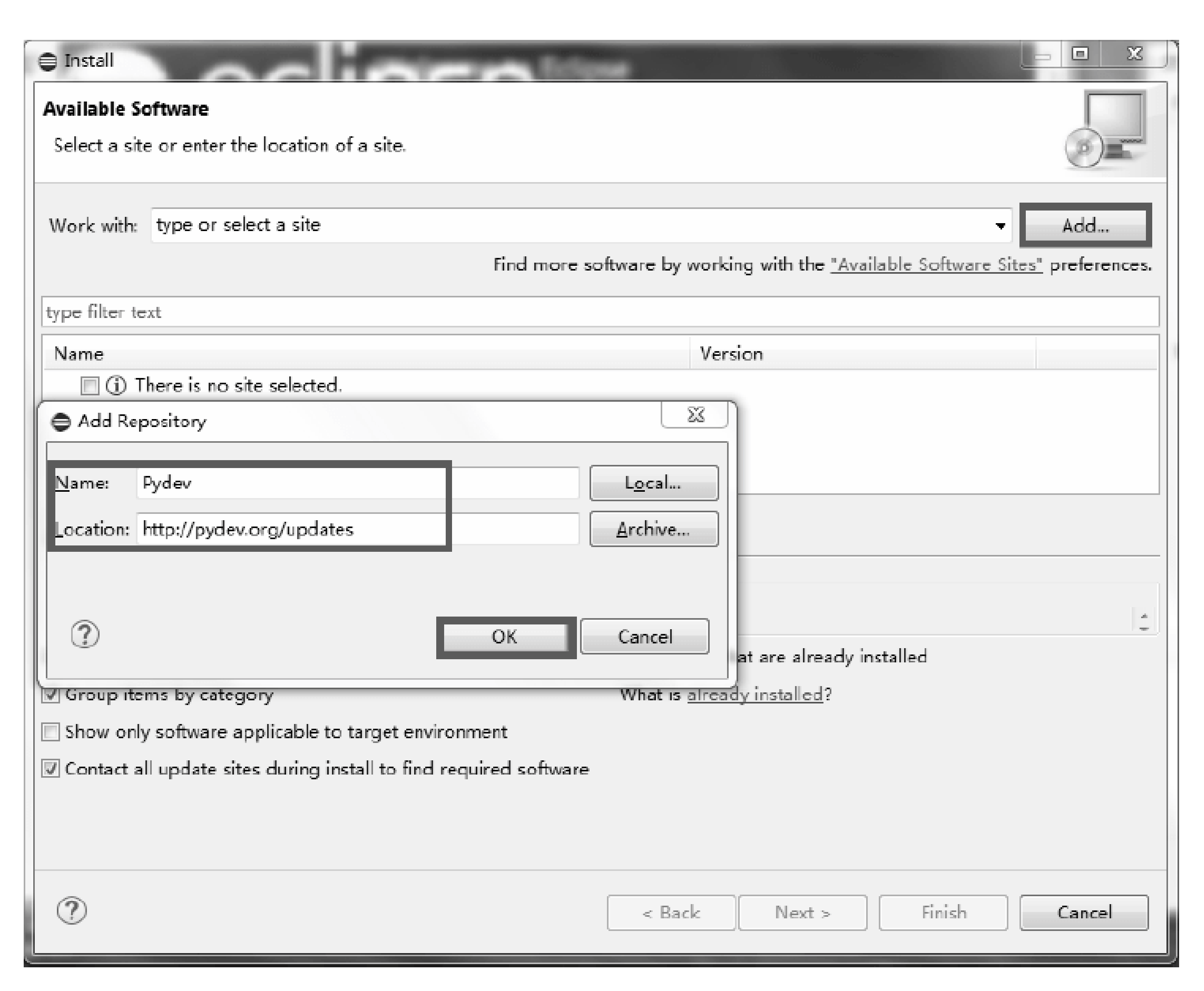


图 6-9 设置 Pydev 安装源

设置完毕后单击 OK 按钮，Eclipse 将显示出这个安装源中所有的可用插件。单击选中 Pydev 插件，单击 Next 按钮，开始安装 Pydev，如图 6-10 所示。

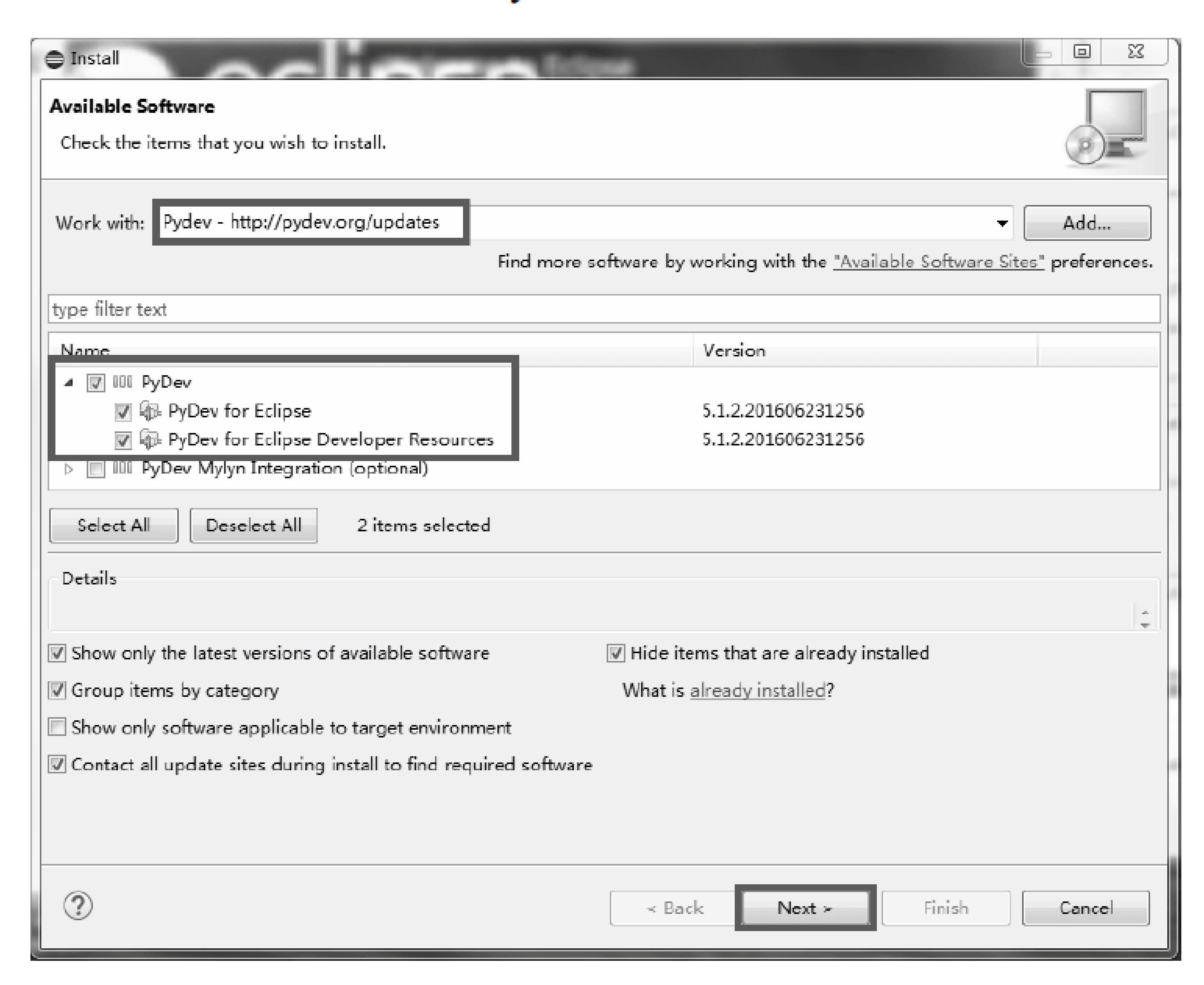


图 6-10 安装 Python 插件 Pydev

单击 Next 按钮，选择同意协议后继续单击 Next 按钮，直到 Pydev 安装完成。因为服务器的缘故，这个安装可能会有点慢。不用着急，Pydev 并不大。如果实在没耐性，也可以用下载工具将 Pydev 先下载到本地，离线安装 Pydev。安装完毕后，按照提示重启 Eclipse，开始配置 Pydev 插件。Pydev 插件只需要配置 Python 解释器的位置就可以使用了，其他的配置

可根据需要自行调试。

在 Eclipse 菜单栏中，单击 Windows 菜单，选择 Preferences 选项。在 Preferences 对话框中单击左侧的 pyDev，选择 Interpreter，单击 Python Interpreter 选项，如图 6-11 所示。

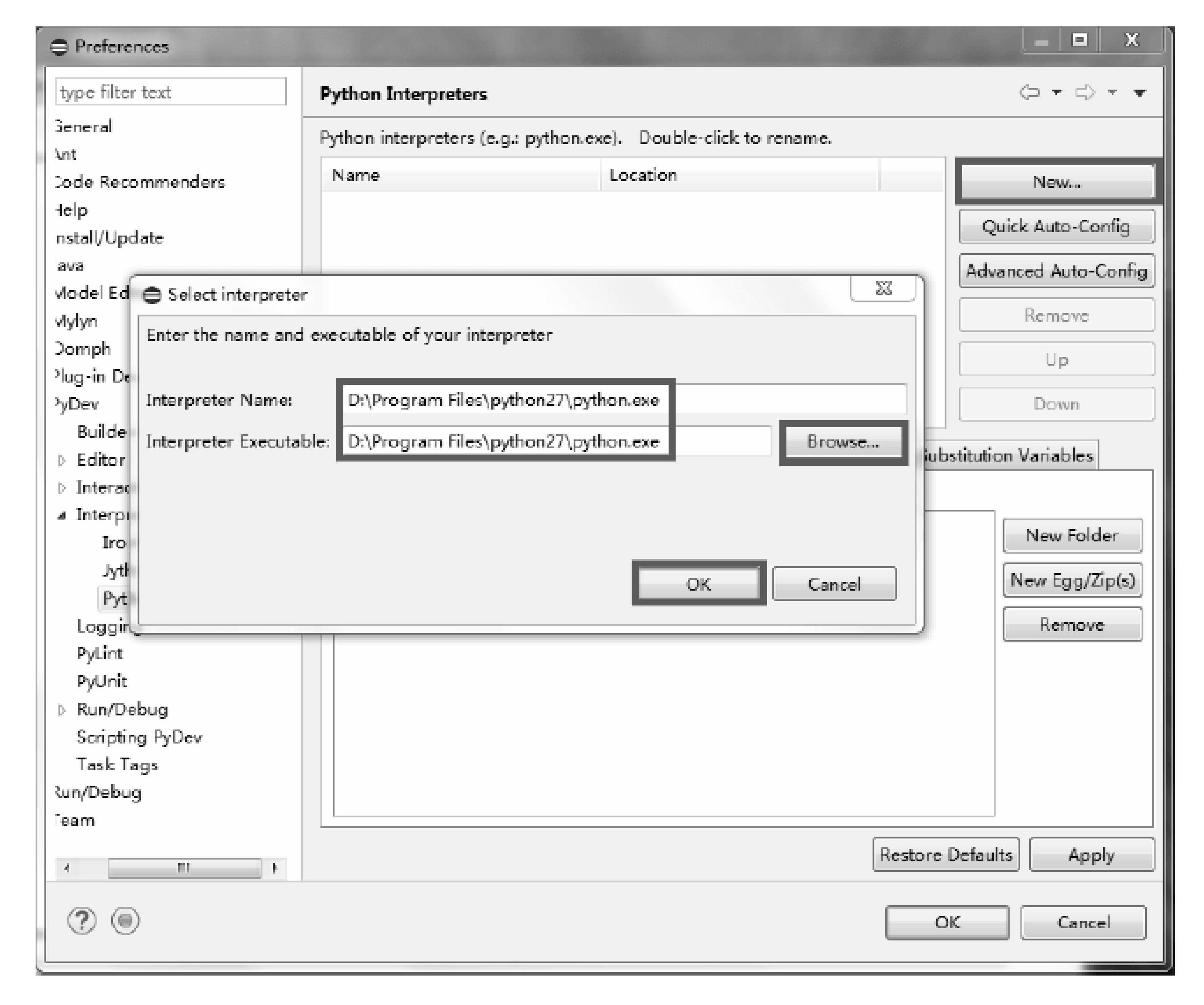


图 6-11 选择 Python 解释器位置

单击 New 按钮，在弹出的对话框中单击 Browse 按钮。选择 python.exe 的路径，单击 OK 按钮直到 Python 解释器导入完毕。一般来说，下一步应该是给 Eclipse 加载中文包。但 Eclipse Neon 版本还很新，中文包并未放出，所以只好暂时使用英文版本的。如果非要中文版的，那只能重新下载低版本的 Eclipse 了。

3．创建 Python 项目

Eclipse 安装配置完毕后，开始创建 Python 项目。打开 Eclipse，单击菜单 File|New|Project 项，创建一个项目，如图 6-12 所示。

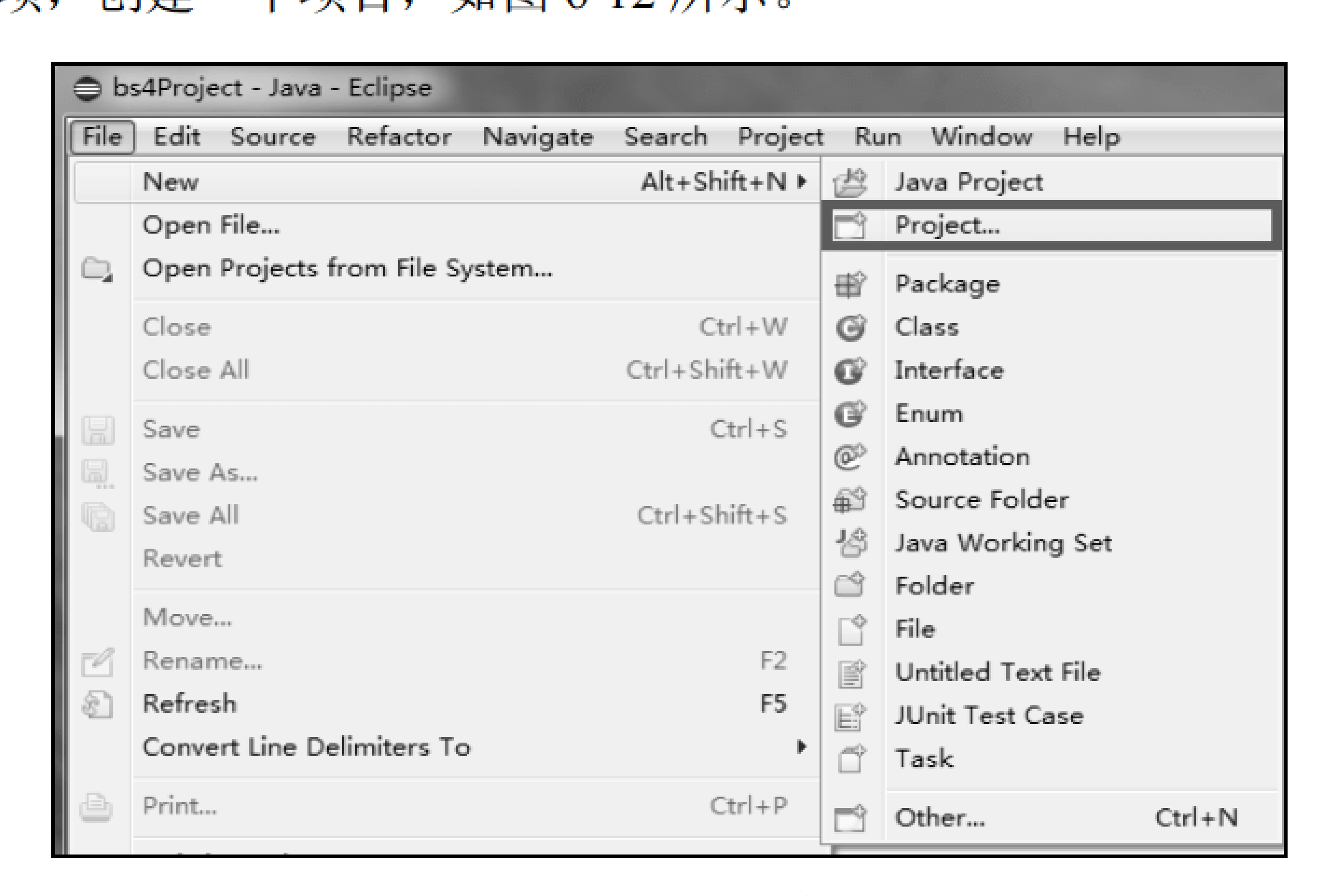


图 6-12 Eclipse 创建项目

在弹出的对话框中选择区项目类型。这里应该选择 PyDev 项目中的 PyDev Project 子项目。选取完毕后，单击 Next 按钮继续，如图 6-13 所示。

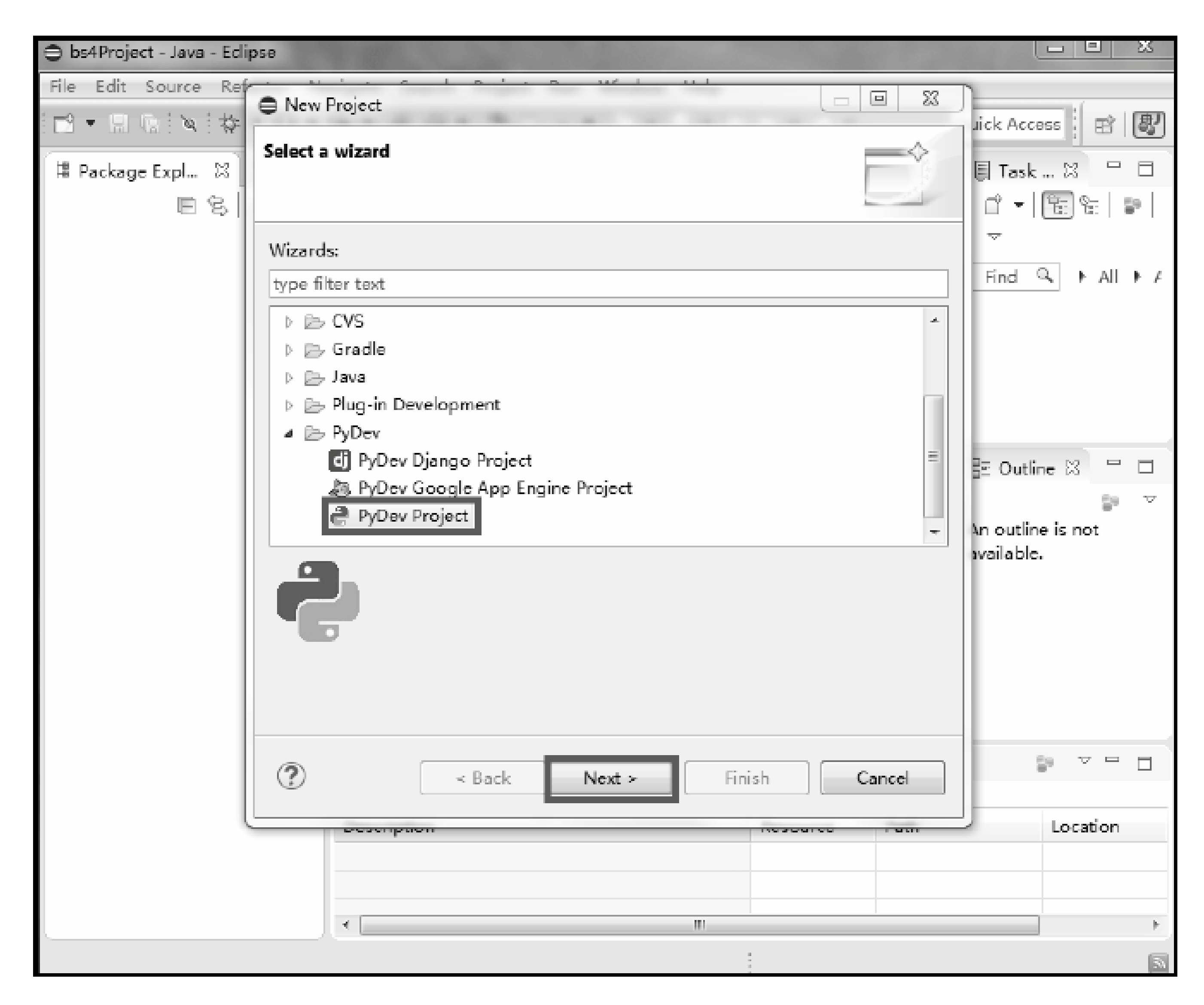


图 6-13　选取项目类型

在弹出的对话框中输入项目名称后，单击 Finish 按钮，项目就创建完毕了，如图 6-14 所示。

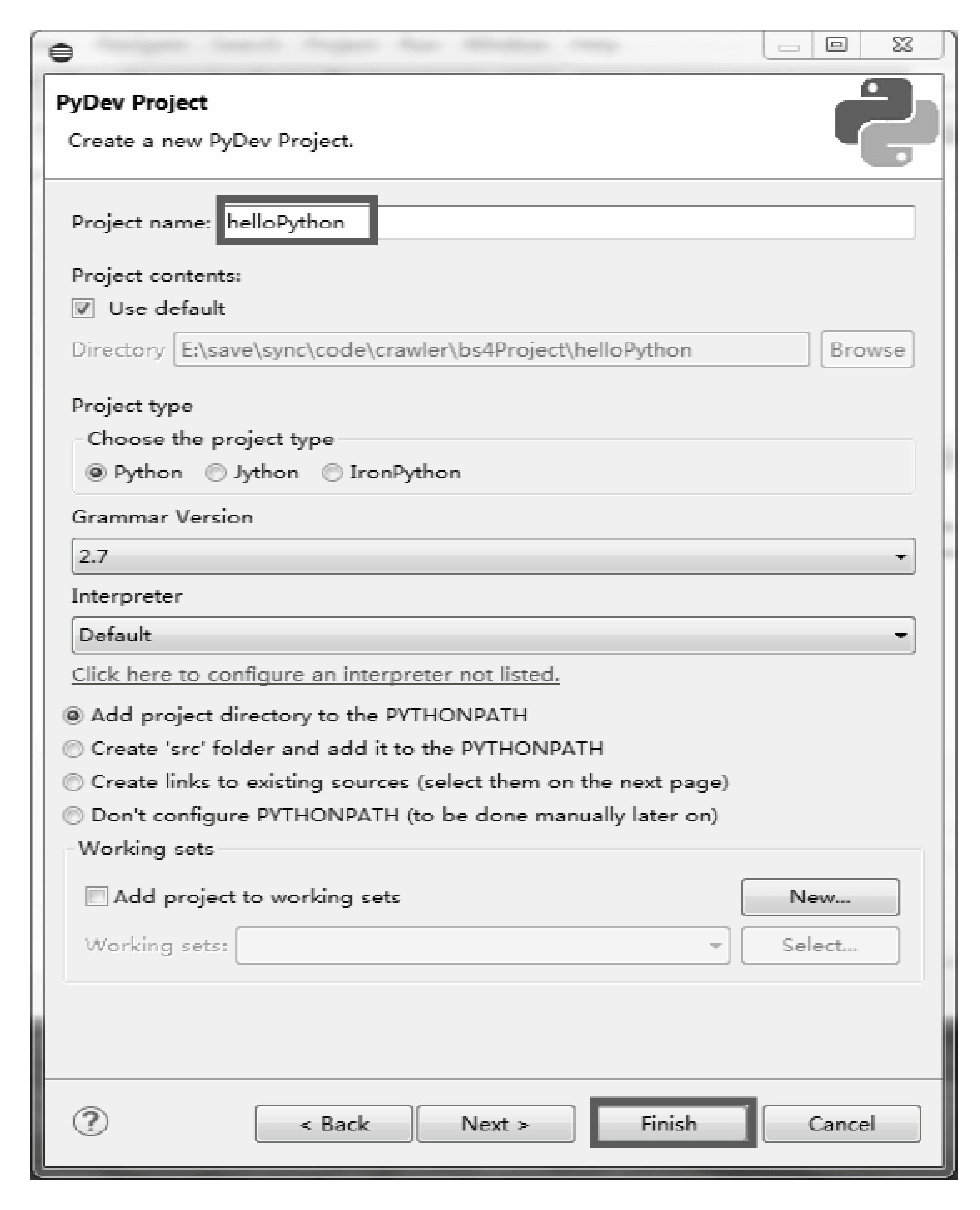


图 6-14　Python 项目名称

回到 Eclipse 的主界面下，在左侧将出现刚创建的 Pydev 项目 helloPython。右击 helloPython 项目将弹出菜单，选择 New 选项，弹出子菜单，如图 6-15 所示。

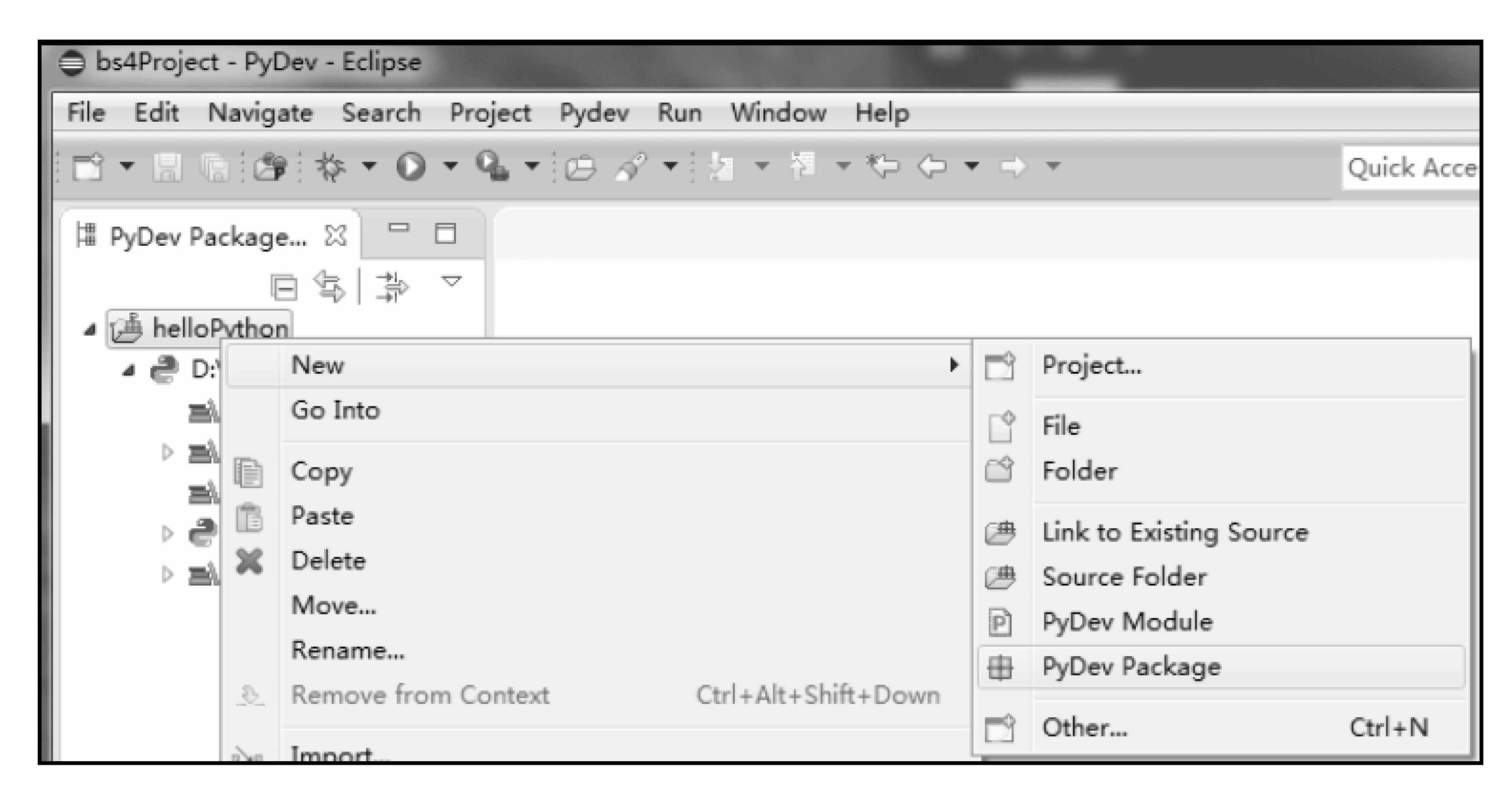


图 6-15　Python 项目创建文件

如果是创建文件和文件夹，正常情况应该是选择 File 和 Folder 选项，但笔者更喜欢使用 PyDev Module 和 PyDev Package 选项。因为选择 File 会创建一个空文件，这个文件里什么都没有。而 PyDev Module 选项会创建一个根据预设模版（菜单 Windows|Preferences|PyDev|Editor|Templates 下的<Empty>项）创建的.py 文件，无须在每次创建文件时再重复设置。Folder 将创建一个空文件夹，而 PyDev Package 将创建一个包含__init__.py 的文件夹（就是在上章中创建的中间件文件夹 middlewares），可以将这个文件夹下的 Python 文件当成模块导入到项目中。

下面来测试一下，创建一个 PyDev Module，名为 hello.py，创建一个 PyDev Package 名为 testModule，并在 testModule 中创建一个 PyDev Module，名为 myModule。创建完毕后的目录结构如图 6-16 所示。

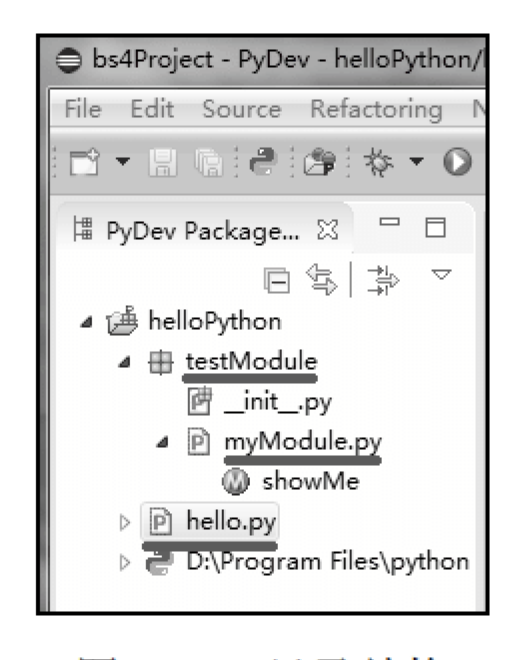


图 6-16　目录结构

【示例 6-1】其中，hello.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on '2016年8月6日'
```

```
@author: hstking hstking@hotmail.com
'''

from testModule.myModule import showMe

if __name__ == '__main__':
    print('Hello, I am first python script on eclipse')
    print('你好，我是在 eclipse 上的第一个 python 程序\n')
    showMe()
```

testModule/myModule.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016 年 8 月 6 日

@author: hstking hstking@hotmail.com
'''

def showMe():
    print('I am a module')
    print('我是一个模块\n')
```

单击 Eclipse 的 Run 菜单，选取 Run 选项（Ctrl + F11），或者直接单击工具栏的 Run 按钮，执行结果如图 6-17 所示。

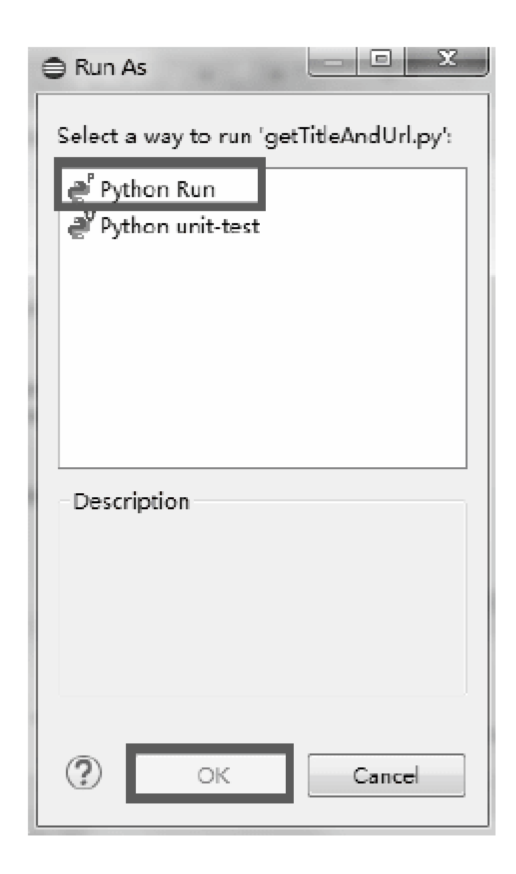


图 6-17　选取 Python 解释器

选择 Python Run 后单击 OK 按钮。运行程序，运行结果如图 6-18 所示。

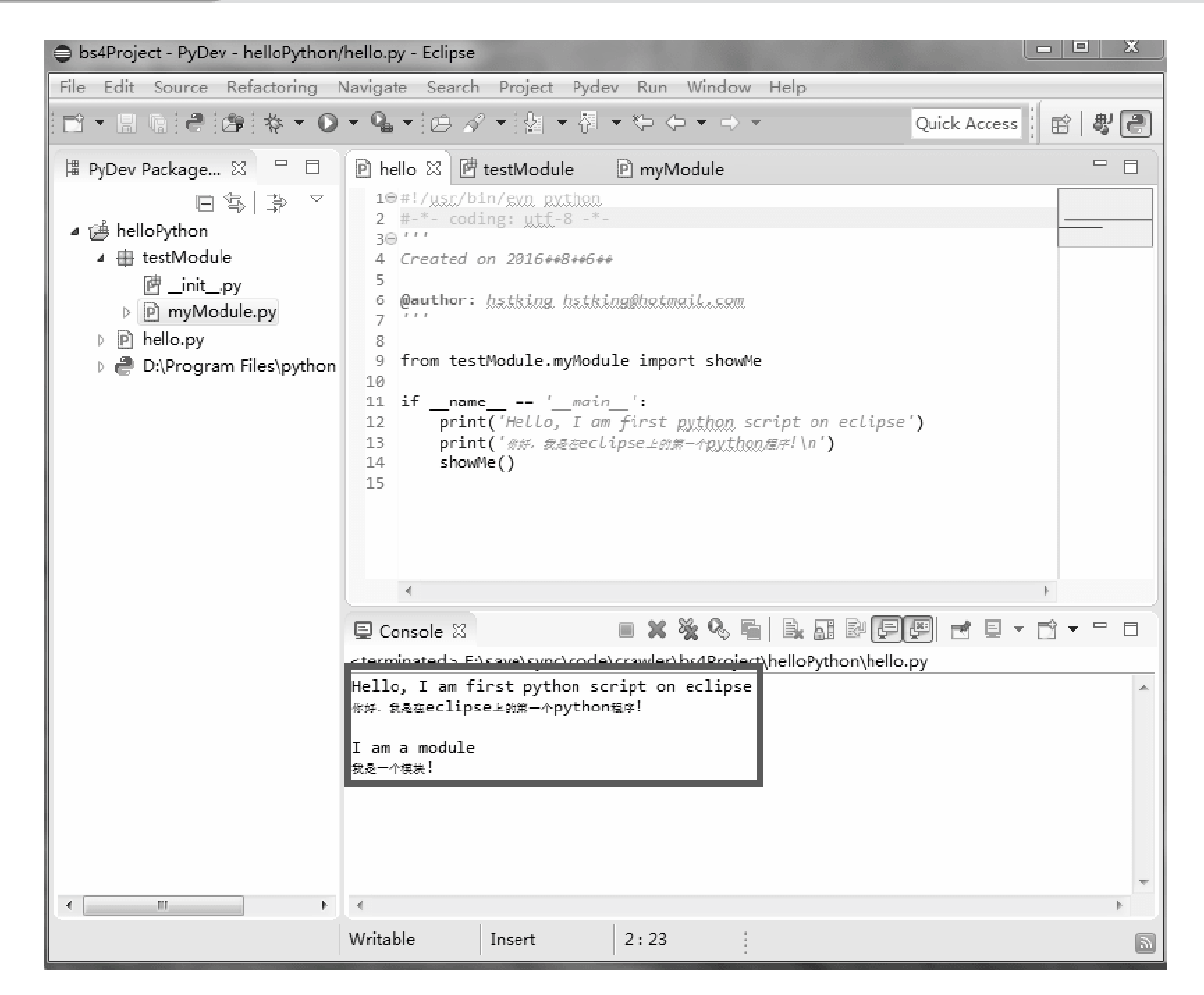


图 6-18　运行 Python 程序

运行无误，Eclipse 测试完毕。选择 Eclipse 做 IDE 除了它跨平台、支持语言丰富外，还因为 Eclipse 有着强大的调试功能。在后面的实例中会演示 Eclipse Debug 调试的强大方便之处。

6.2 BeautifulSoup 解析器

与 Scrapy 相比，bs4 中间多了一道解析的过程（Scrapy 是 URL 返回什么数据，程序就接受什么数据进行过滤）。bs4 则在接收数据和进行过滤之间多了一个解析的过程。根据解析器的不同，最终处理的数据也有所不同。加上这一步骤的优点是可以根据输入数据的不同进行针对式的解析，缺点就是可能会让使用者选择困难，无所适从。在本章中，统一选择 lxml 解析器。

6.2.1　bs4 解析器选择

网络爬虫最终的目的就是过滤选取网络信息，因此最重要的部分就是解析器了。解析器的优劣决定了网络爬虫的速度和效率。Beautiful Soup 除了支持 Python 标准库中的 HTML 解析器外，还支持一些第三方的解析器。表 6-1 中列出了主要的解析器，以及它们的优缺点。

表 6-1　bs4 解析器对比

解析器	使用方法	优点	缺点
Python 标准库	BeautifulSoup(markup,"html.parser")	Python 标准库 执行速度适中 容错能力强	Python 2.7.3 or Python 3.2.2 之前的版本中文容错能力差
lxml HTML 解析器	BeautifulSoup(markup,"lxml")	速度快 容错能力强	需要安装 C 语言库
Lxml XML 解析器	BeautifulSoup(markup,["lxml-xml"]) BeautifulSoup(markup,"xml")	速度快 唯一支持 XML 解析器	需要安装 C 语言库
html5lib	BeautifulSoup(markup,"html5lib")	最好的容错性 以浏览器的方式解析文档 生成 HTML5 格式文档	速度慢 不依赖外部扩展

Beautiful Soup 官方推荐使用 lxml 作为解析器，据说因为 lxml 解析器的效率更高，在此接受官方意见。本章所有的 bs4 爬虫，如无特殊说明都将使用 lxml 解析器。

6.2.2　lxml 解析器安装

1. Windows 下安装 lxml 解析器

先使用 pip 安装试一下，执行命令：

```
pip install lxml
```

执行结果如图 6-19 所示。

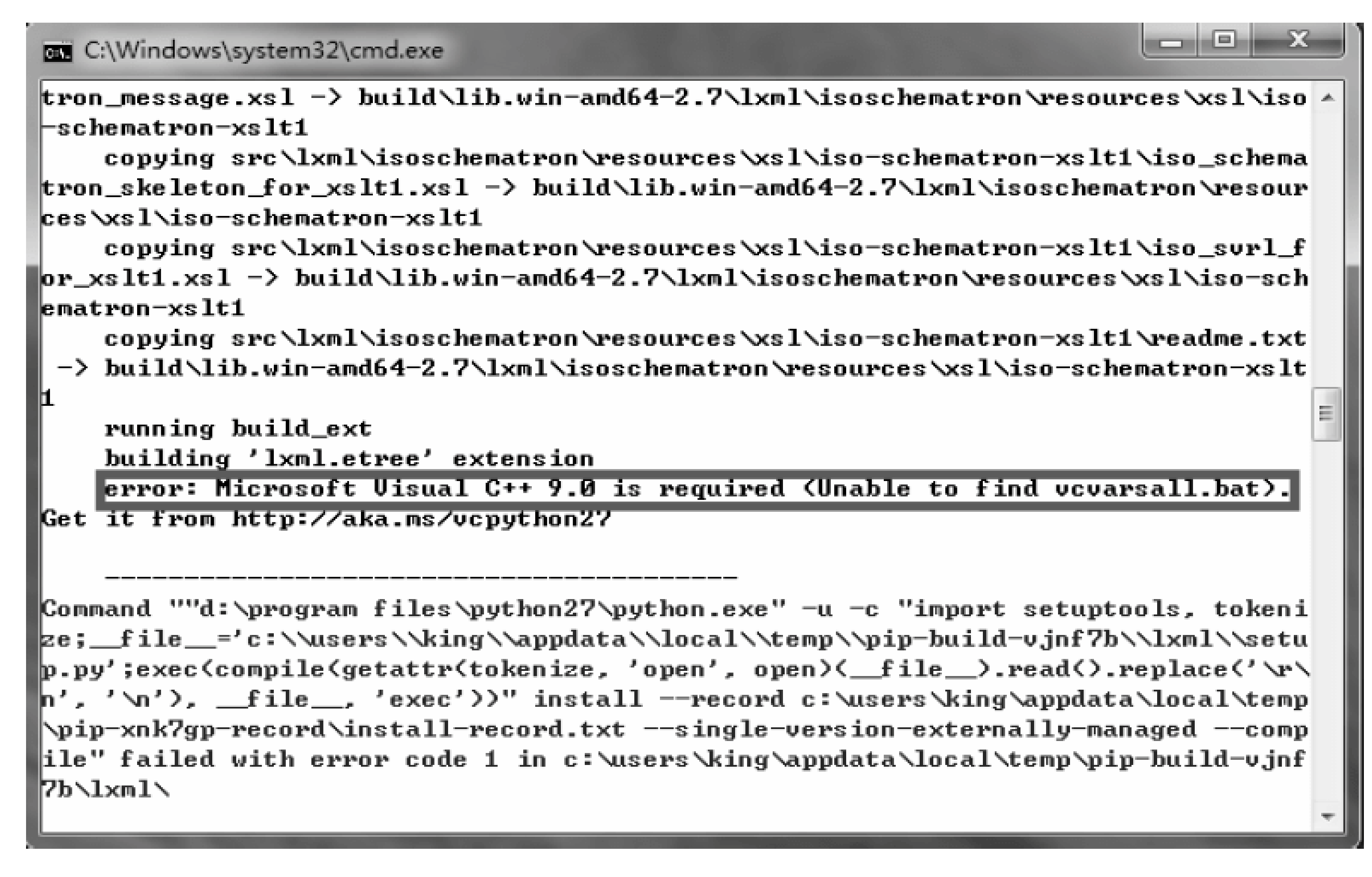


图 6-19　Windows 安装 lxml

在 Windows 下使用 pip 安装 lxml 总会出现这样或那样的问题。如果非要用 pip 安装，那就按照提示的错误一步一步地解决所有的错误，再 pip 安装。如果不追求过程只要求结果，那就简单了。先在 http://www.lfd.uci.edu/~gohlke/pythonlibs/#lxml 下载符合自己系统版本的 lxml（笔者下载的是 lxml-3.6.1-cp27-cp27m-win_amd64.whl），使用 pip 远程安装 wheel 模块后再使用 pip 本地安装 lxml 模块。执行命令：

```
pip install wheel
pip install c:\user\king\Downloads\lxml-3.6.1-cp27-cp27m-win_amd64.whl
```

在 cmd.exe 里测试一下，如图 6-20 所示。

```
C:\Windows\system32\cmd.exe - python
C:\Users\king>python
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import lxml
>>>
>>>
```

图 6-20　测试 lxml 模块

没有提示错误，表明安装成功。

2. Linux 下安装 lxml 解析器

使用 apt-get 安装，执行命令：

```
apt-get install python-lxml
```

执行结果如图 6-21 所示。

```
king@debian: ~
root@debian:/home/king# apt-get install python-lxml
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
建议安装的软件包:
  python-lxml-dbg
下列【新】软件包将被安装:
  python-lxml
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 0 个软件包未被升级
。
需要下载 0 B/754 kB 的软件包。
解压缩后会消耗掉 2,840 kB 的额外空间。
正在选中未选择的软件包 python-lxml。
(正在读取数据库 ... 系统当前共安装有 83305 个文件和目录。)
正准备解包 .../python-lxml_3.4.0-1_amd64.deb  ...
正在解包 python-lxml (3.4.0-1) ...
正在设置 python-lxml (3.4.0-1) ...
root@debian:/home/king#
```

图 6-21　Linux 安装 lxml

显然 Linux 的 apt-get 更加简单方便。

6.2.3　使用 bs4 过滤器

在上一章中，Scrapy 使用 XPath 当过滤器，在网页中过滤得到所需的数据。本章 bs4 则使用 BeautifulSoup 做过滤器。与 XPath 相同的是 BeautifulSoup 同样支持嵌套过滤，可以很

方便地找到数据所在的位置。不同的是 BeautifulSoup 的查找方式更加灵活方便，不但可以通过标签查找，还可通过标签属性来查找。而且 bs4 还可以配合第三方的解析器，可以有针对性地对网页进行解析，使 bs4 威力更加强大、方便。

官网教程上使用的是爱丽丝梦游仙境的内容作为示例文件，但这个文件比较大，看起来没那么直观。这里笔者自建一个 html 的示例文件 scenery.html，通过对 scenery.html 的操作过滤，再对照官网对示例文件的操作方法，很容易就能明白 bs4 是如何过滤提取数据的。

【示例 6-2】自建示例文件 scenery.html 的内容如下：

```
<html>
<head>
   <meta charset="utf-8">
      <title>武汉旅游景点</title>
   <meta name="description" content="武汉旅游景点 精简版" />
   <meta name="author" content="hstking">
</head>
<body>
   <div id="content">
       <div class="title">
          <h3>武汉景点 </h3>
       </div>
       <ul class="table">
          <li>景点 <a>门票价格</a></li>
       </ul>
       <ul class="content">
          <li nu="1">东湖 <a class="price">60 </a></li>
          <li nu="2">磨山 <a class="price">60 </a></li>
          <li nu="3">欢乐谷 <a class="price">108 </a></li>
          <li nu="4">海昌极地海洋世界 <a class="price">150 </a></li>
          <li nu="5">玛雅水上乐园 <a class="price">i50 </a></li>
       </ul>
   </div>
</body>
</html>
```

进入文件目录执行命令：

```
python
from bs4 import BeautifulSoup
soup = BeautifulSoup(open(‘scenery.html’), ‘lxml’)
soup.prettify
```

执行结果如图 6-22 所示。

```
king@debian: ~/code/crawler/bs4Project/seletors
>>> soup.prettify
<bound method BeautifulSoup.prettify of <html>
<head>
<meta charset="utf-8"/>
<title>武汉旅游景点</title>
<meta content="武汉旅游景点 精简版" name="description"/>
<meta content="hstking" name="author"/>
</head>
<body>
<div id="content">
<div class="title">
<h3>武汉景点 </h3>
</div>
<ul class="table">
<li>景点 <a>门票价格</a></li>
</ul>
<ul class="content">
<li nu="1">东湖 <a class="price">60 </a></li>
<li nu="2">磨山 <a class="price">60 </a></li>
<li nu="3">欢乐谷 <a class="price">108 </a></li>
<li nu="4">海昌极地海洋世界 <a class="price">150 </a></li>
<li nu="5">玛雅水上乐园 <a class="price">150 </a></li>
</ul>
</div>
```

图 6-22　soup.prettify



bs4 会将所有输入的内容的字符编码编为 unicode。输入内容为英文时还看不出什么优势，但在过滤中文网页时会非常方便。

一个文件或一个网页，在导入 BeautifulSoup 处理之前，bs4 并不知道它的字符编码是什么。在导入 BeautifulSoup 过程中，它会自动地猜测这个文件或是网页的字符编码。常用的编码当然会又快又好地猜出来。但不常用的编码呢？好在 BeautifulSoup 还有两个非常重要的参数：exclude_encoding 和 from_encoding。

参数 exclude_encoding 的作用是排除掉不正确的字符编码。例如，已经非常确定网页不是 iso-8859-7 也不是 gb2312 编码，但又未知网页编码时，就可以使用命令：

```
    soup = BeaurtifulSoup(response.read(), exclude_encoding=['iso-8859-
7','gb2312'])
```

此时 bs4 就会放弃猜测这两种编码。比如如果已知网页的具体编码是 big5，也可以直接使用 from_encoding 参数确定编码，让 bs4 放弃猜测，可以使用命令：

```
    soup = BeaurtifulSoup(response.read(), exclude_encoding='big5')
```

一般来说 bs4 都不需要自己确定编码，常用的字符编码它都能检测出来。但有时碰见比较生僻的编码时，这两个命令就显得非常重要了。中文的字符编码是个非常讨厌的问题，如果不知道文件的字符编码，而 bs4 又解析编码错误时，那就只有根据官网的方法，安装 chardet 或者 cchardet 模块，然后使用 UnicodeDammit 自动检测了。

解决字符编码这个问题后，已经得到了 soup 这个 bs4 的类。在 soup 中，bs4 将网页节点解析成了一个个 Tag。同名的 Tag（HTML 中的标签就那么几个，所以同名的 Tag 会非常多）会有不同的属性。即使同名又同属性的 Tag，它们又有顺序和父标签的区别。bs4 就是通过这些不同将所需的数据过滤出来的。

这里的 Tag 与 html 或 xml 中的 Tag 是一致的。执行命令：

```
Tag1 = soup.ul
```

得到的结果如图 6-23 所示。

以上命令的作用是通过 soup 来获取第一次出现的标签名为 ul 的标签内容。用同样的方法，可以获取第一个出现的，标签名为 div、li、head、title……的标签内容。

如果某个标签只出现了一次，比如<head>、<title>标签，通常只会出现一次。可以用 soup.head 和 soup.title 的方式获取标签内容，还可以用 bs4 过滤器 soup.find(Tag, [attrs])的方法获取第一次出现的标签的内容。执行命令：

```
soup.ul
soup.find('ul')
```

执行结果如图 6-24 所示。

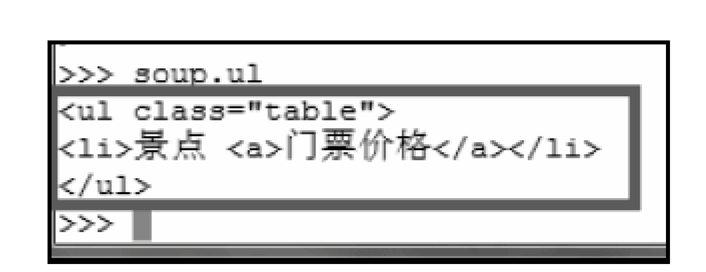


图 6-23　soup 获取 Tag

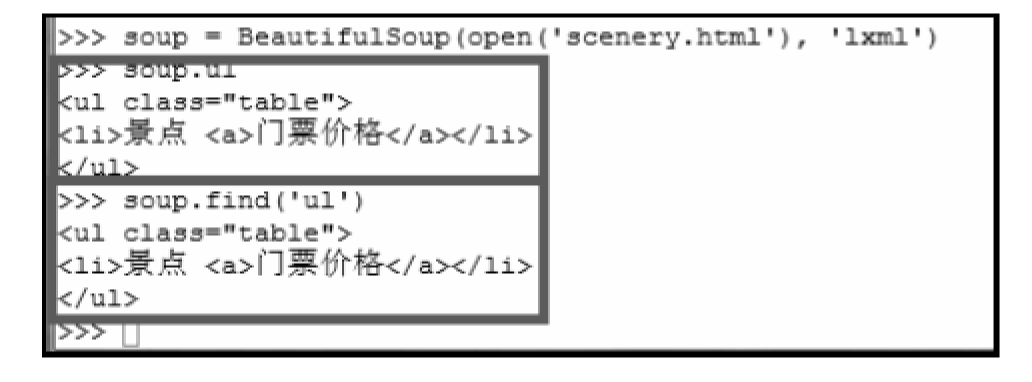


图 6-24　soup.find 获取 Tag

在一个 HTML 文件中，有的标签肯定不止出现一次。具体到这个示例文件 scenery.html 中，<div>、<ul>、<li>就不止出现一次。第一次出现的标签位置如何确定已经很清楚了，那第二次、第三次、第 N 次呢？bs4 给出的方法是 soup.find_all(Tag, [attrs])。使用 soup.find_all 命令可以获取所有符合条件的标签列表，然后直接从列表中读取就可以了。执行命令：

```
soup.find_all('ul')
soup.find_all('ul')[0]
soup.find_all('ul')[1]
```

执行结果如图 6-25 所示。

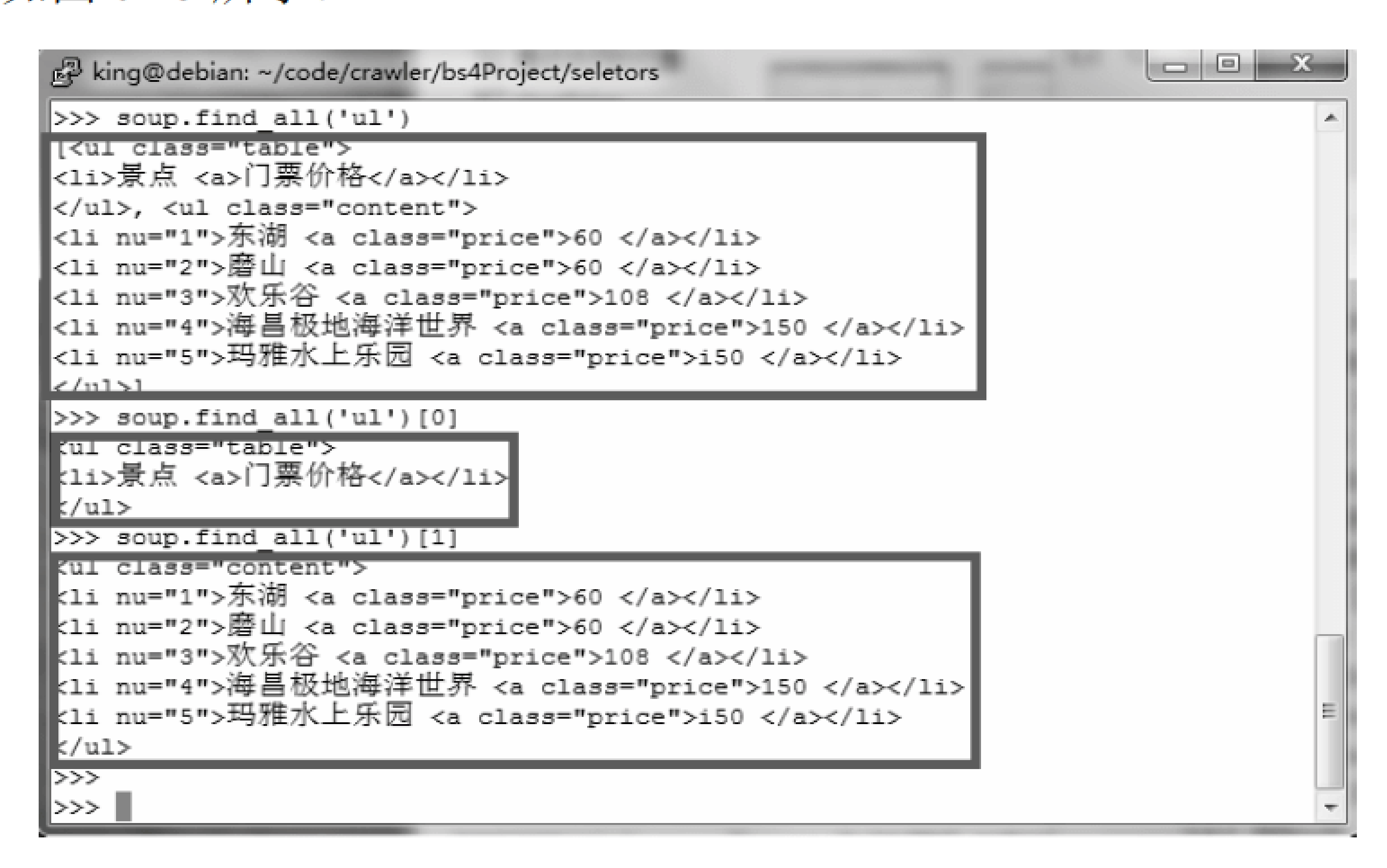


图 6-25　soup.find_all 获取 Tag

从顺序上来区别同名标签，这样出现再多的同名标签也可以很从容地定位了。在 html 中，同名标签比较少时，可以先用 soup.find_all 来获取标签位置列表，再用一个个数的方法来确定标签位置。如果这个列表比较短还好，从 1 数到 20 还可以接受。如果这个列表很长呢？从 1 数到 100 那就太讨厌了。

还是以 scenery.html 文件为例，文件中的<li>标签，目前在文件中只出现了 6 次。如果列出所有的景点，标签<li>出现 60 次都不奇怪。仔细观察一下 li 标签，它们除了名字相同外，还有一个相同的属性 nu，而属性 nu 的值是不同的。bs4 过滤器 Soup.find 和 soup.find_all 都支持名字+属性值定位。

如果一个 html 文件中出现了标签名相同，属性不同的标签。例如，scenery.html 文件中的<li>标签，可以用 soup.find(TagName, attrs={attrName:attrValue})的方法获取 Tag 的位置。比如需要定位标签文字为欢乐谷的那个<li>标签（<li>标签的属性是相同的，都是 nu，只是属性值不同）。可以执行命令：

```
soup.find('li', attrs={'nu':'3'})
```

执行结果如图 6-26 所示。

```
>>> soup.find('li', attrs={'nu':'3'})
<li nu="3">欢乐谷 <a class="price">108 </a></li>
>>>
```

图 6-26　soup.find 配合属性获取标签

如果标签名相同，属性相同，连属性值都相同的标签，那就用 soup.find_all(tagName, attrs={‘attName’:’attValue’})将所有符合条件的标签装入列表，然后从列表中慢慢地数。请放心，一般情况下，标签名相同，属性相同，连属性值都相同的标签在任何一个文件中都是很少见的。在 scenery.html 文件中，符合这个条件的只有<a>标签（如果是显示完整的景点<a>标签也会很多，那是下一步的问题）。如果需要获取景点“磨山”这一行的<a>标签，可以执行命令：

```
Tags = soup.find_all('a', attrs={'class':'price'})
Tags[1]
```

执行结果如图 6-27 所示。

```
>>> Tags = soup.find_all('a', attrs={'class':'price'})
>>> Tags[1]
<a class="price">60 </a>
>>>
```

图 6-27　soup.find_all 配合属性获取标签

目前 HTML 没有列出所有的景点，<a>标签还比较少，如果列出了所有景点<a>标签很多，该怎么办？再仔细观察一下<a>标签，所有<a>标签的标签名、属性和属性值虽然是相同的，但它们的上级标签（也就是常说的父标签）的标签名、属性和属性值总不可能相同吧？即使运气再差点，上级标签的标签名、属性、属性值都相同，上上级标签的难道也相同？还

是以获取景点，“磨山”这一行的<a>标签为例，执行命令：

```
tmpTag = soup.find('li', attrs={'nu':'2'})
tmpTag.a
tmpTag.find('a')
```

执行结果如图 6-28 所示。

```
>>> Tags = soup.find_all('a', attrs={'class':'price'})
>>> Tags[1]
<a class="price">60 </a>
>>>
>>> tmpTag = soup.find('li', attrs={'nu':'2'})
>>> tmpTag.a
<a class="price">60 </a>
>>>
>>> tmpTag.find('a')
<a class="price">60 </a>
>>>
```

图 6-28　soup 嵌套获取标签

这种不直接定位目标标签，先间接定位目标标签的上级（也可以是下级）标签，再间接定位目标标签的方法，有点类似于 Scrapy 中 XPath 的嵌套过滤了。

实际上，如果觉得目标标签没什么显著特征，上级标签和下级标签也没有什么显著特征。还可以定位目标标签的兄弟标签。不过这种方法一般很少用，这里就不多说了。如果有兴趣可以参考官方文档。

一般来说，最终需要获取保存的数据都不会是标签，而是标签里的数据，这个数据有可能是标签所包含的字符串，也有可能是标签的属性值。不过获取标签后，要数据那就很简单了。例如获取示例文件中海昌极地海洋世界这个景点的序号和票价，也就是这一行标签<li>的属性 nu 的值和<a>标签包含的字符串。可以执行命令：

```
Tag = soup.find('li', attrs={'nu':'4'})
Tag.get(‘nu’)
Tag.a.get_text()’
```

执行结果如图 6-29 所示。

```
>>> Tag = soup.find('li', attrs={'nu':'4'})
>>> Tag
<li nu="4">海昌极地海洋世界 <a class="price">150 </a></li>
>>> Tag.get('nu')
'4'
>>> Tag.a.get_text()
u'150 '
>>> Tag.find('a').get_text()
u'150 '
>>>
```

图 6-29　soup 获取数据

如果只需要做简单爬虫，了解以上知识就可以了。如果需要对 bs4 进行深度挖掘，还需要读者自行参考 bs4 的官方文档。

6.3 bs4 爬虫实战一：获取百度贴吧内容

笔者是个美剧迷，经常在网上追剧，偶尔也在百度贴吧上看看美剧贴，可懒癌又比较严重，天天登录贴吧查看贴子又觉得很麻烦。干脆就写个爬虫让它自动爬内容好了，有空就看看哪些帖子回复了，又有哪些新贴。这里以百度贴吧里的“权利的游戏吧”为例。

6.3.1 目标分析

百度贴吧中“权利的游戏”的 URL 是 http://tieba.baidu.com/f?kw=%E6%9D%83%E5%88%A9%E7%9A%84%E6%B8%B8%E6%88%8F&ie=utf-8&pn=0。看起来很乱是不是？仔细看看这个 URL，其中包含有 ie=utf-8，说明那个浏览器接受的是 utf8 的字符编码，而在 Windows 下默认的是 GBK 码，所以在 Windwos 下如果想让浏览器能正常识别 URL，就必须把 Windows 默认的 GBK 转换成 utf8 格式。URL 是纯英文的情况下，转不转换都没关系，反正都是一样的。当 URL 含有中文时，转换后的 URL 就会出现乱码，如图 6-30 所示。

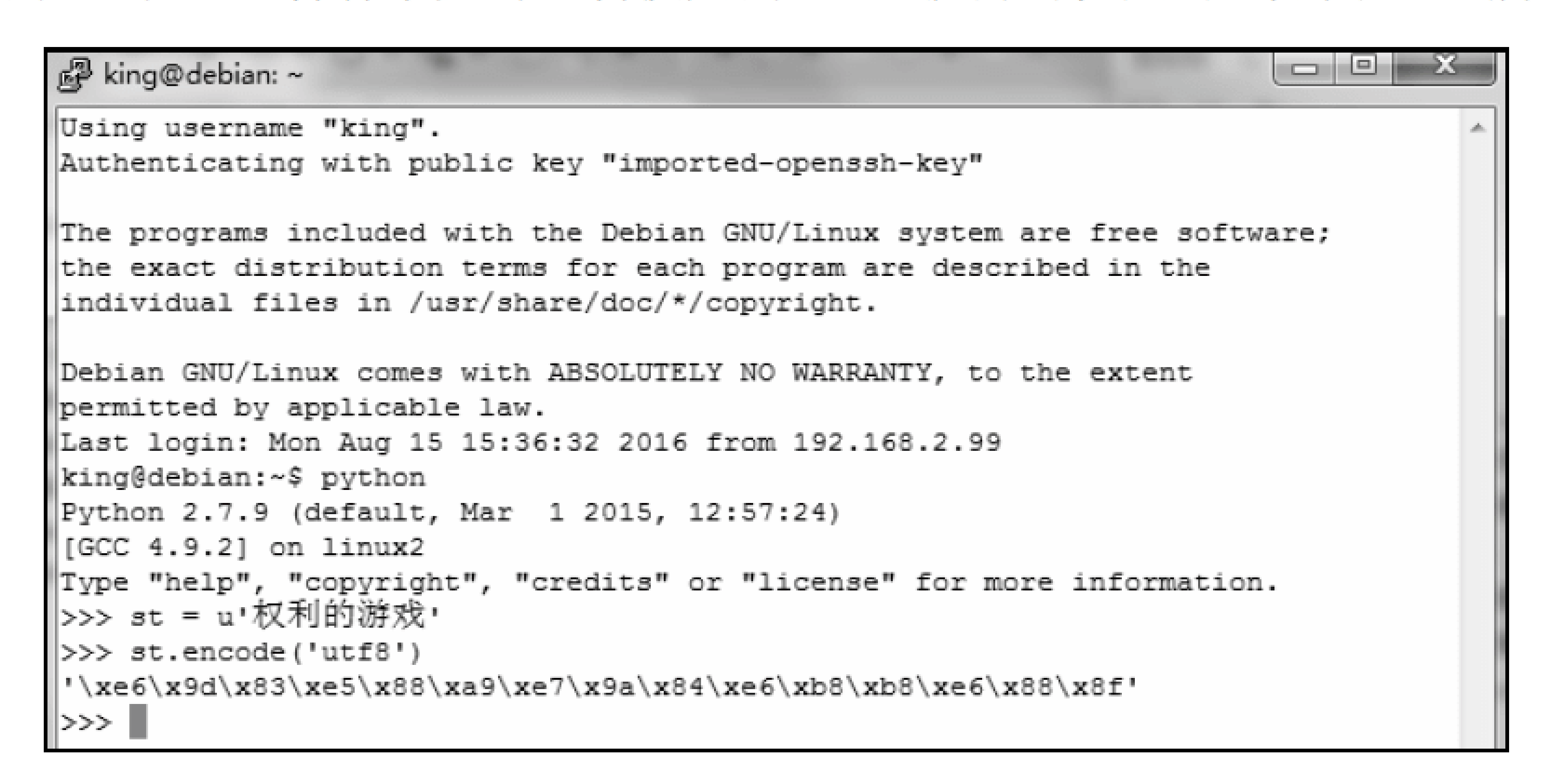


图 6-30　编码转换

这个由“权利的游戏”转换而来的乱码是不是很眼熟。所以这个 URL 原本的状态应该是 http://tieba.baidu.com/f?kw=权利的游戏&ie=utf-8&pn=0，在交给浏览器或者 python 程序处理时再将它转换成 utf8 编码就可以了。

在网页上单击“下一页”按钮，浏览器跳转得到的 URL 是 http://tieba.baidu.com/f?kw=%E6%9D%83%E5%88%A9%E7%9A%84%E6%B8%B8%E6%88%8F&ie=utf-8&pn=50。好的，明白了。每按一次“下一页”按钮，pn 将增加 50。

在浏览器（这里使用的是 Chrome 浏览器，其他浏览器基本上都差不多）中打开这个 URL（utf8 版的和 GBK 版本的都行，浏览器会自动转换成 utf8 版的，但在 python 程序中必须使用 utf8 版的），查看帖子标题，如图 6-31 所示。



图 6-31　bs4 爬虫来源页面

在页面空白处右击选择“查看页面源代码”，在页面源代码网页按 Ctrl+F 键打开查找框，在查找框内输入第一个帖子的标题名，找到所需数据位置，如图 6-32 所示。

```
<ul id="thread list" class="threadlist bright j_threadlist_bright">     权利的游戏1-6季，无修剪来   第 1 条，共 2 条
    <li class=" j_thread_list clearfix" data-
field='{&quot;id&quot;:4673258358,&quot;author_name&quot;:&quot;Carpe__Diem__&quot;,&quot;first_post_id&quot;:93981199839,&quot;repl
:374,&quot;is_bakan&quot;:null,&quot;vid&quot;:&quot;&quot;,&quot;is_good&quot;:null,&quot;is_top&quot;:null,&quot;is_protal&quot;:nu
_membertop&quot;:null,&quot;frs_tpoint&quot;:null}' >
            <div class="t_con cleafix">

                    <div class="col2_left j_threadlist_li_left">
                <span class="threadlist_rep_num center_text"
                    title="回复">374</span>
            </div>
                <div class="col2_right j_threadlist_li_right ">
            <div class="threadlist_lz clearfix">
                <div class="threadlist_title pull_left j_th_tit
">


    <a href="/p/4673258358" title="权利的游戏1-6季，无修剪来" target="_blank" class="j_th_tit ">权利的游戏1-6季，无修剪来</a>
</div><div class="threadlist_author pull_right >
```

图 6-32　所需数据位置

找到所需数据的位置后，仔细观察一下，发现所有帖子都有一个共同的标签<li class=" j_thread_list clearfix">（这个标签还有其他的属性，只需要一个共同的属性就够了）。所以只需要用 bs4 过滤器 find_all 找到所有的标签，然后再进一步分离出所需的数据就可以了。

6.3.2　项目实施

既然思路都已经明确了，那就开工吧。打开 Eclipse，单击 New 图标右侧的三角按钮，在弹出菜单中选择 PyDev Project 项，如图 6-33 所示。

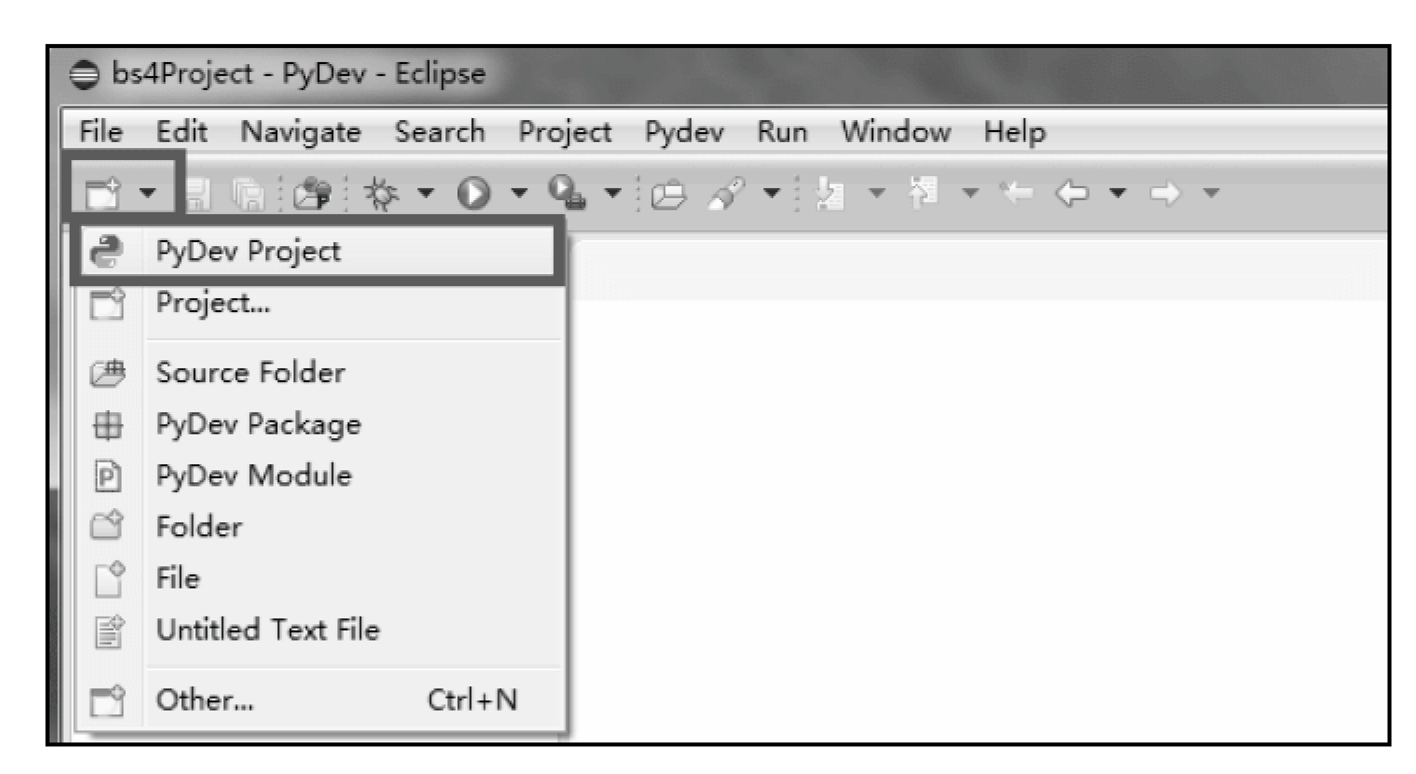


图 6-33　Eclipse 创建 Python 新项目

在弹出的对话框中输入项目名称，单击 Finish 按钮，如图 6-34 所示。

图 6-34　输入项目名

在 Eclipse 的左侧右击刚建立的项目 baiduBS4，在弹出的菜单中选择 New|PyDev Module 菜单。在项目中创建一个新的 Python 模块（前面提到过，这里选择 PyDev Module 是为了方便。非要选择 file，从零开始一步步地创建一个 Python 文件当然也可以），如图 6-35 所示。

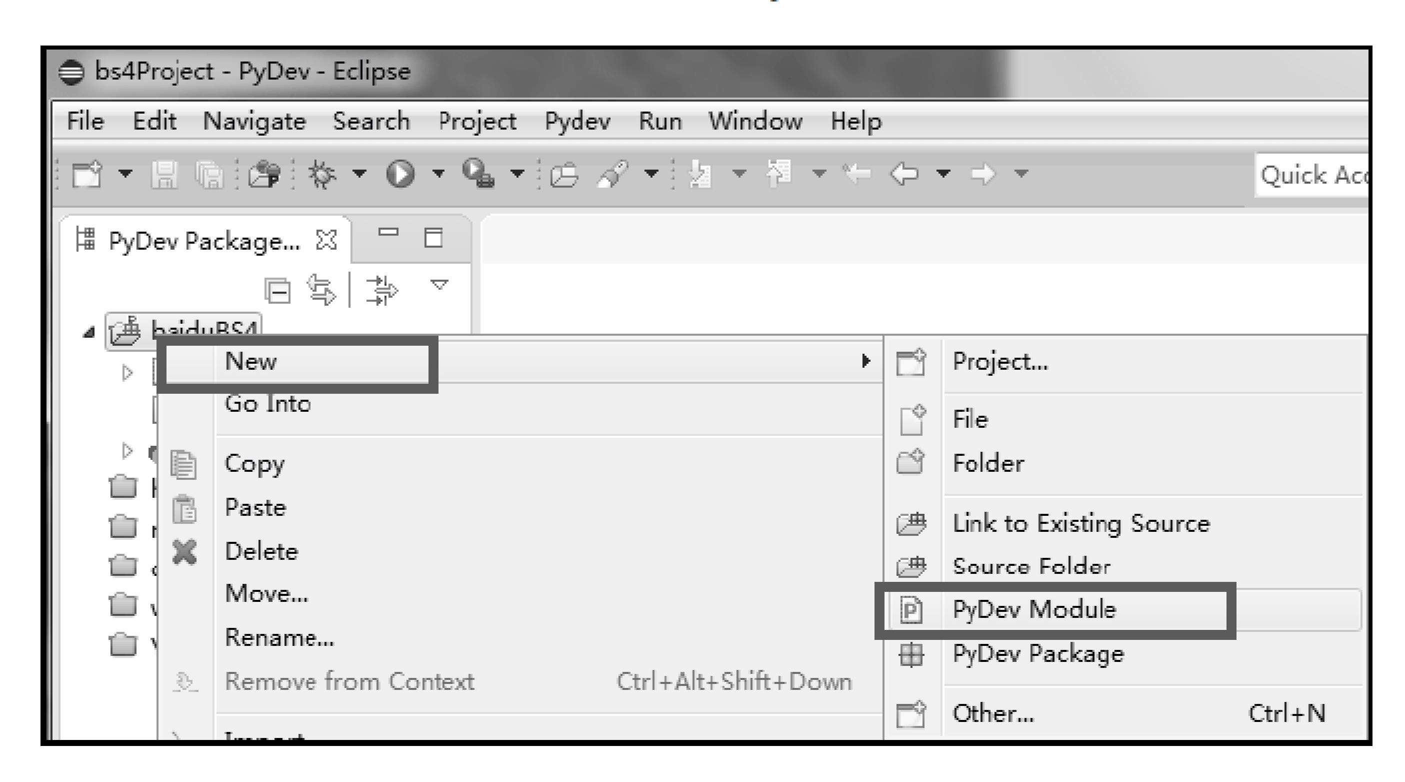


图 6-35　项目中创建 Python 文件

在弹出的对话框中输入 Python 文件的文件名（无须加后缀名），右击 Finish 按钮。Python 文件创建完成，如图 6-36 所示。

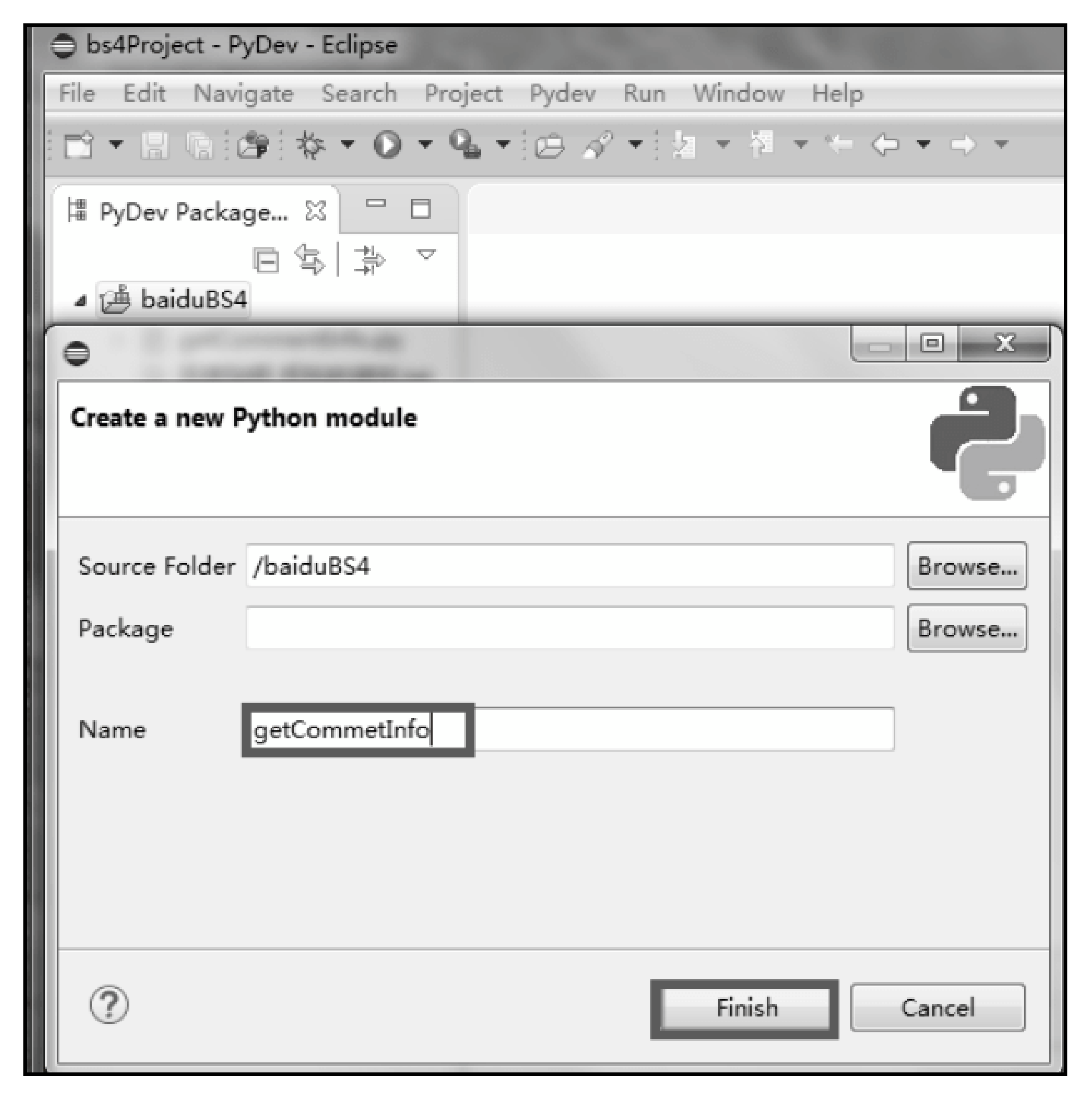


图 6-36　Python 文件名

【示例 6-3】在新创建的 getCommentInfo.py 中输入代码，getCommentInfo.py 的代码如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016 年 8 月 9 日

@author: hstking hstking@hotmail.com
'''


import urllib2
from bs4 import BeautifulSoup
from mylog import MyLog as mylog


class Item(object):
    title = None    #帖子标题
    firstAuthor = None  #帖子创建者
    firstTime = None  #帖子创建时间
    reNum = None    #总回复数
    content = None  #最后回复内容
```

```
    lastAuthor = None   #最后回复者
    lastTime = None #最后回复时间


class GetTiebaInfo(object):
    def __init__(self,url):
        self.url = url
        self.log = mylog()
        self.pageSum = 5
        self.urls = self.getUrls(self.pageSum)
        self.items = self.spider(self.urls)
        self.pipelines(self.items)

    def getUrls(self,pageSum):
        urls = []
        pns = [str(i*50) for i in range(pageSum)]
        ul = self.url.split('=')
        for pn in pns:
            ul[-1] = pn
            url = '='.join(ul)
            urls.append(url)
        self.log.info(u'获取 URLS 成功')
        return urls

    def spider(self, urls):
        items = []
        for url in urls:
            htmlContent = self.getResponseContent(url)
            soup = BeautifulSoup(htmlContent, 'lxml')
            tagsli = soup.find_all('li',attrs={'class':' j_thread_list clearfix'})
            for tag in tagsli:
                item = Item()
                item.title = tag.find('a', attrs={'class':'j_th_tit'}).get_text().strip()
                item.firstAuthor = tag.find('span', attrs={'class':'frs-author-name-wrap'}).a.get_text().strip()
                item.firstTime = tag.find('span', attrs={'title':u'创建时间'.encode('utf8')}).get_text().strip()
                item.reNum = tag.find('span', attrs={'title':u'回复'.encode('utf8')}).get_text().strip()
                item.content = tag.find('div', attrs={'class':'threadlist_abs threadlist_abs_onlyline '}).get_text().strip()
```

```
                item.lastAuthor = tag.find('span', attrs={'class':'tb_icon_author_rely j_replyer'}).a.get_text().strip()
                item.lastTime = tag.find('span', attrs={'title':u'最后回复时间'.encode('utf8')}).get_text().strip()
                items.append(item)
                self.log.info(u'获取标题为<<%s>>的项成功 ...' %item.title)
        return items

    def pipelines(self, items):
        fileName = u'百度贴吧_权利的游戏.txt'.encode('GBK')
        with open(fileName, 'w') as fp:
            for item in items:
                fp.write('title:%s \t author:%s \t firstTime:%s \n content:%s \n return:%s \n lastAuthor:%s \t lastTime:%s \n\n\n\n'
%(item.title.encode('utf8'),item.firstAuthor.encode('utf8'),item.firstTime.encode('utf8'),item.content.encode('utf8'),item.reNum.encode('utf8'),item.lastAuthor.encode('utf8'),item.lastTime.encode('utf8')))
                self.log.info(u'标题为<<%s>>的项输入到"%s"成功' %(item.title, fileName.decode('GBK')))

    def getResponseContent(self, url):
        '''这里单独使用一个函数返回页面返回值，是为了后期方便加入 proxy 和 headers 等
        '''
        try:
            response = urllib2.urlopen(url.encode('utf8'))
        except:
            self.log.error(u'Python 返回 URL:%s  数据失败' %url)
        else:
            self.log.info(u'Python 返回 URUL:%s  数据成功' %url)
            return response.read()


if __name__ == '__main__':
    url = u'http://tieba.baidu.com/f?kw=权利的游戏&ie=utf-8&pn=50'
    GTI = GetTiebaInfo(url)
```

按照上面的步骤，在项目中重新建立一个名为 mylog.py 的 PyDev Module（也可以是一个单纯的 File）。

【示例 6-4】mylog.py 的内容如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

```
#Author  :hstking
#E-mail  :hstking@hotmail.com
#Ctime   :2015/09/15
#Mtime   :
#Version :

import logging
import getpass
import sys


#### 定义MyLog类
class MyLog(object):
#### 类MyLog的构造函数
    def __init__(self):
        self.user = getpass.getuser()
        self.logger = logging.getLogger(self.user)
        self.logger.setLevel(logging.DEBUG)

####  日志文件名
        self.logFile = sys.argv[0][0:-3] + '.log'
        self.formatter = logging.Formatter('%(asctime)-12s %(levelname)-8s %(name)-10s %(message)-12s\r\n')

####  日志显示到屏幕上并输出到日志文件内
        self.logHand = logging.FileHandler(self.logFile, encoding='utf8')
        self.logHand.setFormatter(self.formatter)
        self.logHand.setLevel(logging.DEBUG)

        self.logHandSt = logging.StreamHandler()
        self.logHandSt.setFormatter(self.formatter)
        self.logHandSt.setLevel(logging.DEBUG)

        self.logger.addHandler(self.logHand)
        self.logger.addHandler(self.logHandSt)

####  日志的5个级别对应以下的5个函数
    def debug(self,msg):
        self.logger.debug(msg)

    def info(self,msg):
        self.logger.info(msg)

```

```
    def warn(self,msg):
        self.logger.warn(msg)

    def error(self,msg):
        self.logger.error(msg)

    def critical(self,msg):
        self.logger.critical(msg)

if __name__ == '__main__':
    mylog = MyLog()
    mylog.debug(u"I'm debug 测试中文")
    mylog.info("I'm info")
    mylog.warn("I'm warn")
    mylog.error(u"I'm error 测试中文")
    mylog.critical("I'm critical")
```

项目代码已经完成了。此时 Eclipse 中的项目如图 6-37 所示。

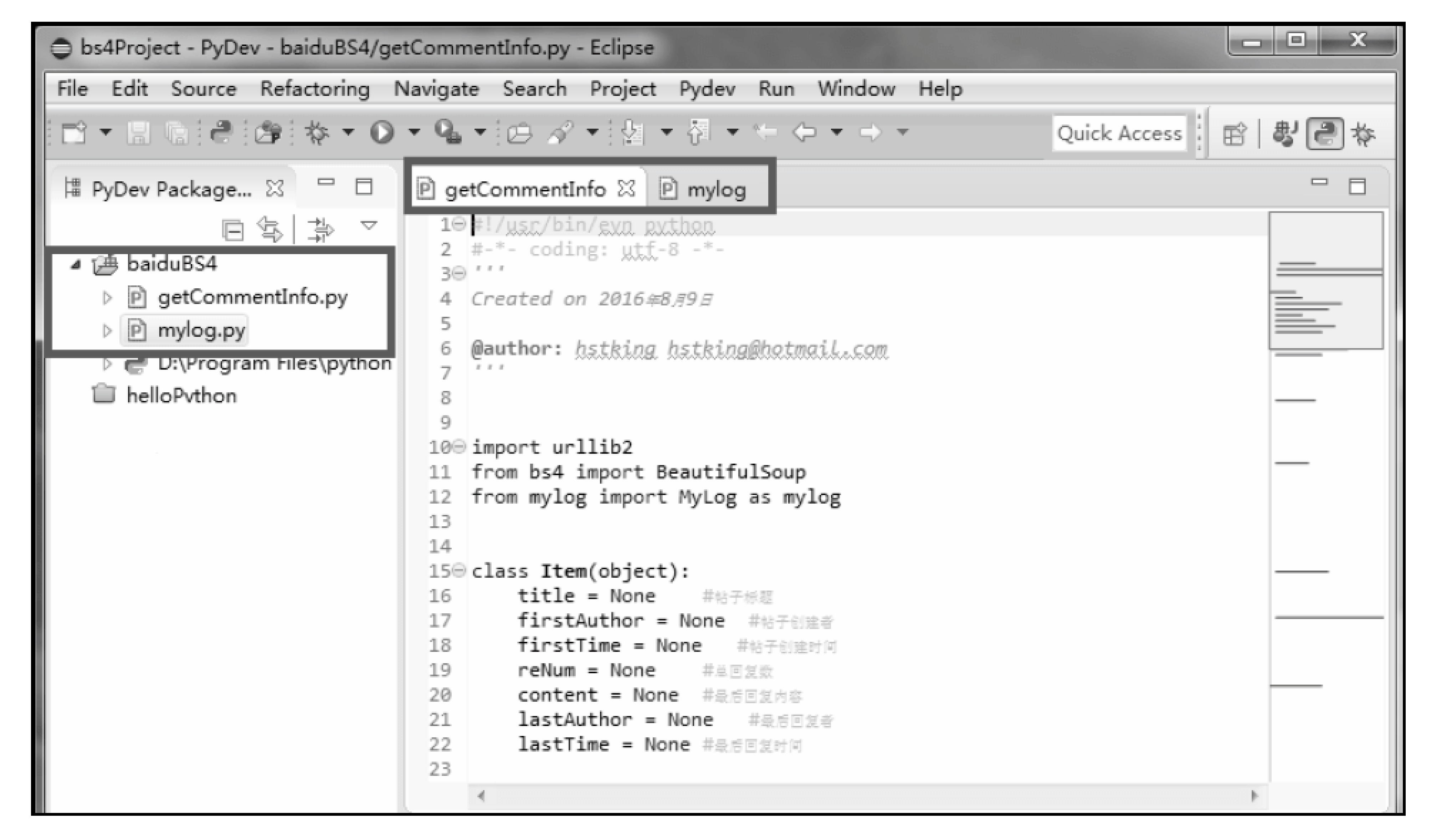


图 6-37　Eclipse 项目 baiduBS4

单击菜单栏上的运行图标，程序运行完毕后单击 baiduBS4 项目图标，按 F5 键刷新。得到的结果如图 6-38 所示。

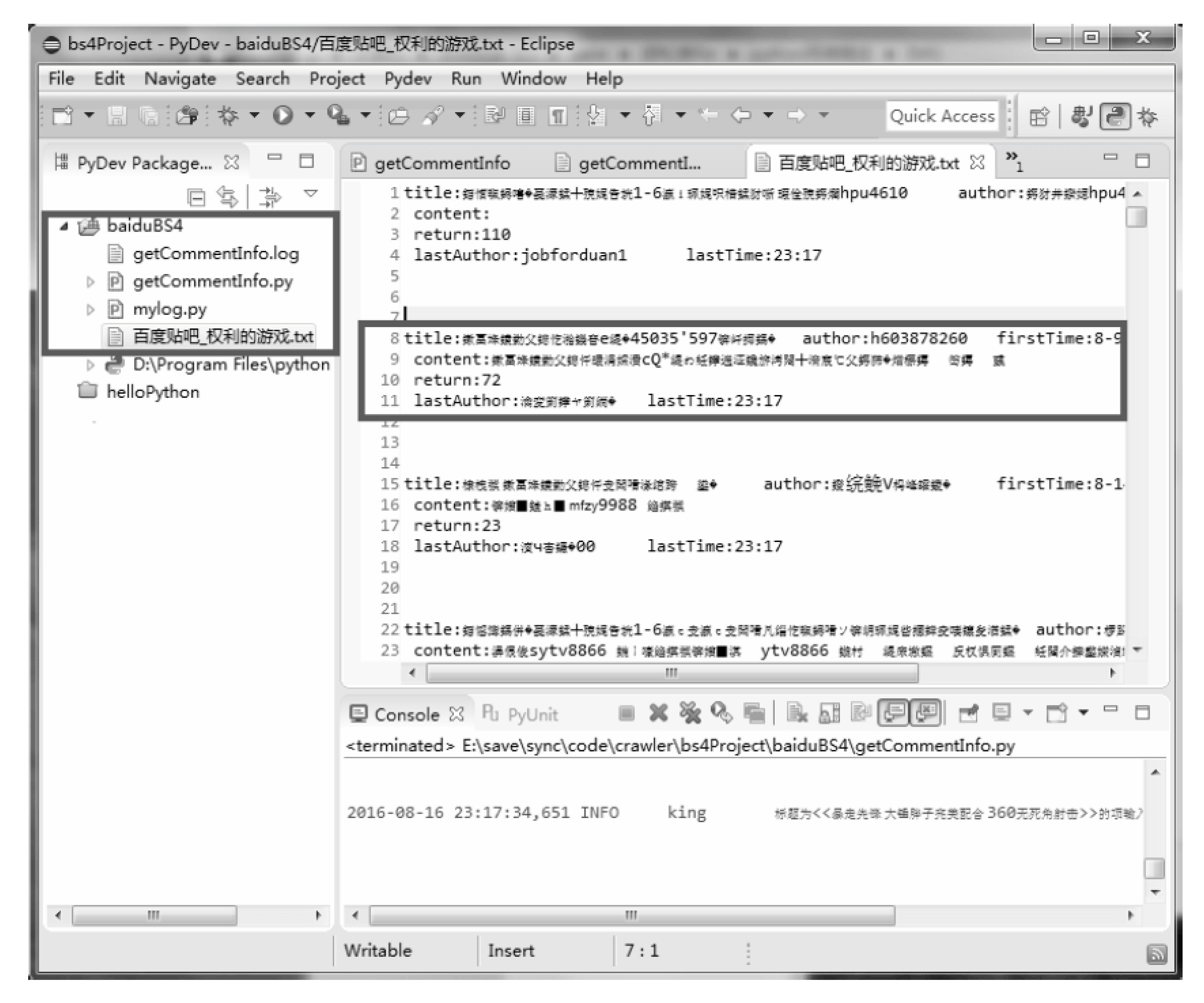


图 6-38　Eclipse 运行结果

运行完毕后得到了两个新文件。一个是 log 文件 getCommentInfo.log，一个是爬虫保存结果文件“百度贴吧_权利的游戏.txt”。在“百度贴吧_权利的游戏.txt”文件中的中文明显有乱码，没关系，那是因为直接用 Eclipse 的编辑器打开的缘故。右击 Eclipse 左边“百度贴吧_权利的游戏.txt”的图标，在弹出的菜单中选择 Open With 菜单，然后在弹出的子菜单中选择 System Eitor 项，使用 Windows 自带的笔记本打开文件，乱码就不见了，如图 6-39 所示。

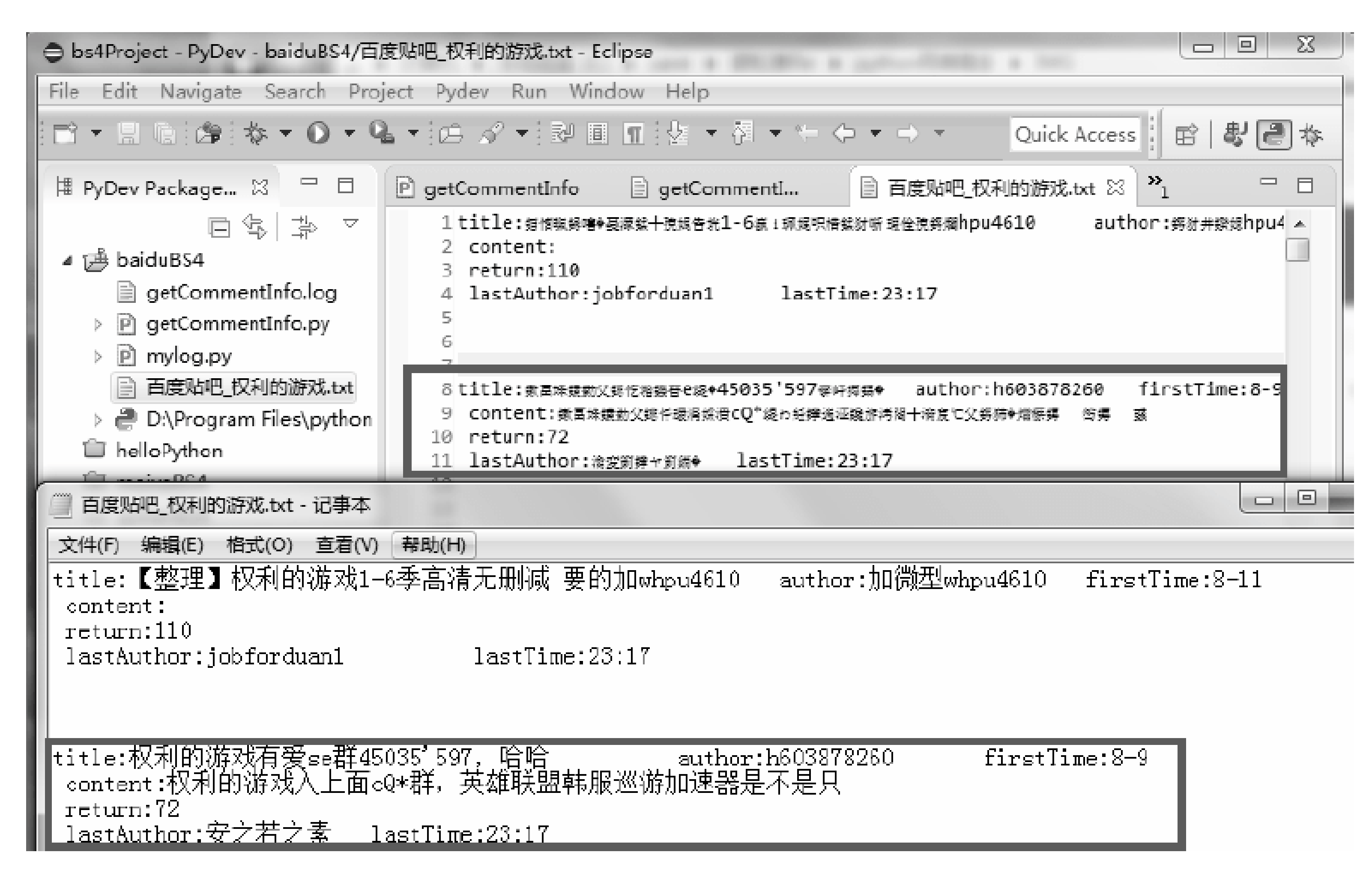


图 6-39　笔记本打开文件

这是因为“百度贴吧_权利的游戏.txt”文件中数据保存的是 utf8 编码。Eclipse 自带的文

字编辑器默认支持的是 GBK 编码，所以会显示乱码。而 Windows 记事本 notepad 虽然也默认支持的是 GBK，但是它同时也支持 utf8 编码。

6.3.3　代码分析

在项目 baiduBS4 中除了主程序外，笔者还自定义了一个 mylog.py 模块（对，就是模块，这个 python 程序就是做模块用的）。这个模块的作用很明显，就是为了主程序提供 log 功能。

log 功能很重要。虽然 Eclipse 已经提供了非常方便的 Debug 功能。没有 log 配合就只能从头到尾一步一步地调试。几步十几步那也就忍了，可爬虫动辄几十页上百页的爬行，一旦出错，没有 log 帮助定位，很难找到错误点。

这个 mylog.py 写得很简单。只是将 Python 的标准模块 logging 简单地包装了一下。第 9~11 行导入了所需的 Python 模块。第 15 行创建了一个新的类。第 18~24 行定义了 log 文件的文件名、用户名、log 的等级以及 log 文件的格式。这里要稍做说明的是，在 log 的格式 self.formatter 的最后添加了一个\r\n。那是因为在 Windows 下换行符号是\r\n，如果是在 Linux 下，加\n 就可以了。mylog.py 这个自建模块在 Windows 和 Linux 下基本是通用的。

第 27~36 行则定义了两个 loghandler。一个是将 log 输出到文本中，一个是将 log 输出到终端方便调试。第 39~52 行则按照 logging 模块定义了 5 个 log 级别。

主程序 getCommentInfo.py 也比较简单。第 10~12 行还是导入所需的模块。在第 15~22 行定义了一个新类。还记得 Scrapy 框架中的 items.py 吗？主程序里的 Item 类就是仿照 Scrapy 框架中的 items.py 写的。个人认为 Scrapy 的框架非常方便，也很合理，Scrapy 优秀的地方就直接学习借鉴了。也可以完全参考 Scrapy 的方法，重新建立一个 Python 模块，将这个类放到一个单独的文件中。

第 25 行创建了一个爬虫类。第 27 行定义了爬虫的入口 URL。第 28 行为类创建了一个 log。第 29 行则定义了爬行的页数，这里定义只爬行了 5 页，实际上有接近 1000 页可爬。如有必要，完全可以一网打尽。第 30~32 行执行类函数。

第 34~43 行定义了一个 getUrls 的类函数。这个函数的作用是根据页面变化的规律（每向后翻一页，pn 增加 50），将所有的 URL 装入一个列表中去，供下一个类函数调用。

第 45~62 行定义了一个 spider 的类函数。这个函数也参考了 Scrapy 的 spider。与 scrapy 不同的是 Scrapy 使用的是 XPath 过滤器，而这里使用的是 bs4 的 BeautifulSoup 过滤有用数据。第 50 行使用 soup.find_all 函数将所有符合条件的数据装入了 tagsli 列表中。第 51~61 行使用 for 函数遍历整个列表，将有效数据过滤出来，添加到 items 列表中，供下一个函数处理。

第 64~70 行把得到的 items 列表写入到最终的数据保存文件“百度贴吧_权利的游戏.txt”中去。这里要注意的是第 65 行，因为 Win7 默认的是 GBK 编码，而文件名中又含有中文字符，所以只有先用 u 字符将字符串 unicode 化，再用 encode(‘GBK’)将整个字符串转换成 GBK 编码。

第 72~81 行完全可以精简成一句，直接返回 urllib2.urlopen(url).read()就可以了。这里用单独一个函数是为了以后扩充功能。比如使用 proxy 代理，添加 headers 等。这个就有点类似于 Scrapy 框架中的中间件。一旦发现爬虫被禁止，就可以在这个函数中做出相应的修改，避开封锁。这里要注意的是需要将 URL 字符串转换成 utf8 编码。

第 84~86 行是__main__函数，实例化 GetTiebaInfo 类。

6.3.4 Eclipse 调试

虽然在 Windows 下也可以使用 pdb 模块对 Python 程序进行调试。Pdb 的强大当然是无须质疑，但直观上来说就远远不如 Eclipse 了。下面就牛刀小试，实验一下 Eclipse 的调试功能。

首先要做的是在程序中添加断点，也就是程序运行中暂停中断的位置。在所需中断行的最前方，行号的前面空白处双击右键，会出现一个绿色气球的图标，表明断点已经设立成功。这里只在 41 行和 60 行设置了 2 个断点，如图 6-40 所示。

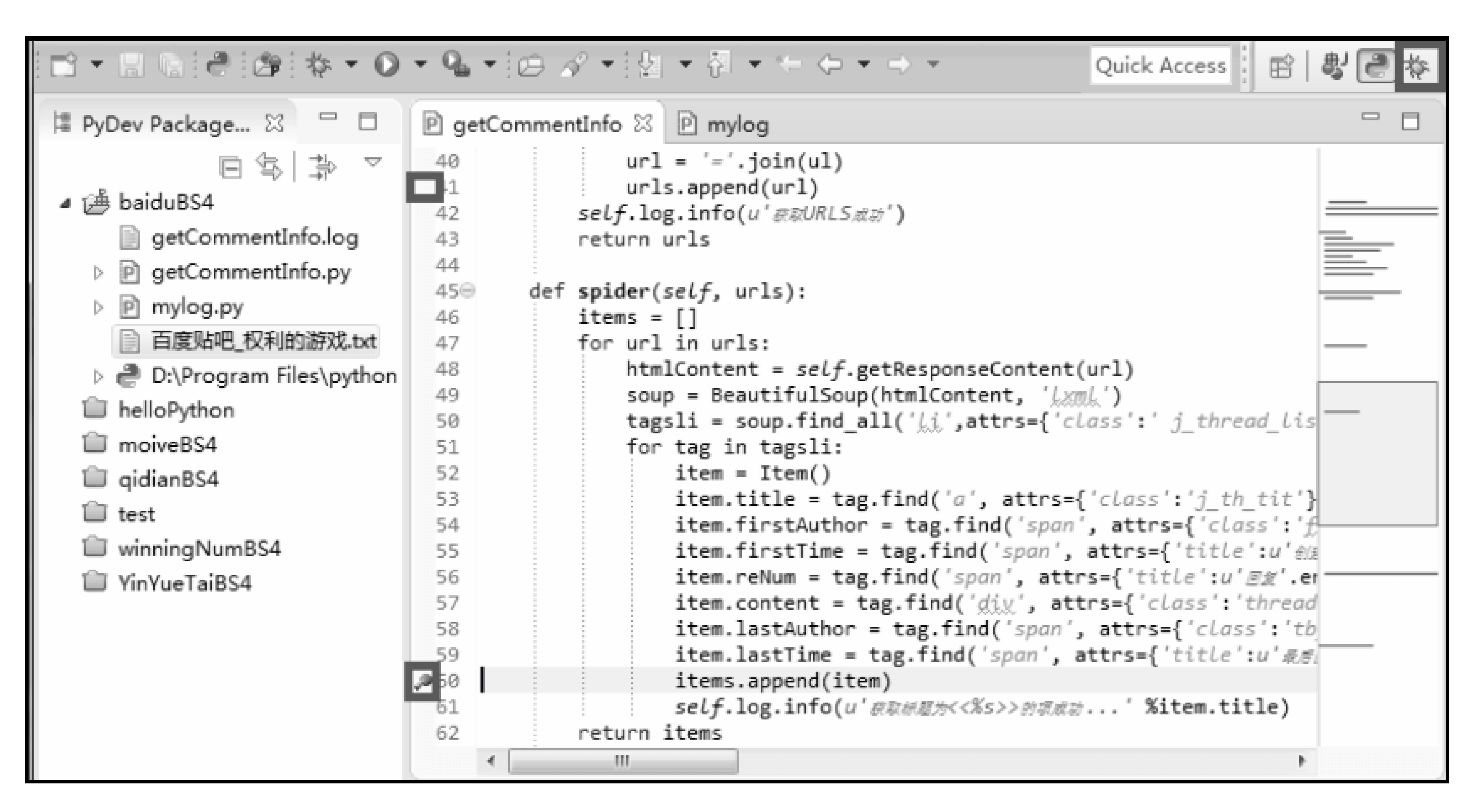


图 6-40　Eclipse 设置断点

再单击 Eclipse 上方菜单栏最右边的爬虫图标。进入调试模式。单击图标栏的爬虫图标开始调试。程序运行到断点处会自动停止运行，单击图标栏的箭头图标进行单步调试。在程序栏中的箭头指向运行的位置。可以在变量标签中观察变量的值。测试完毕后可以单击图标栏的停止图标退出调试，如图 6-41 所示。

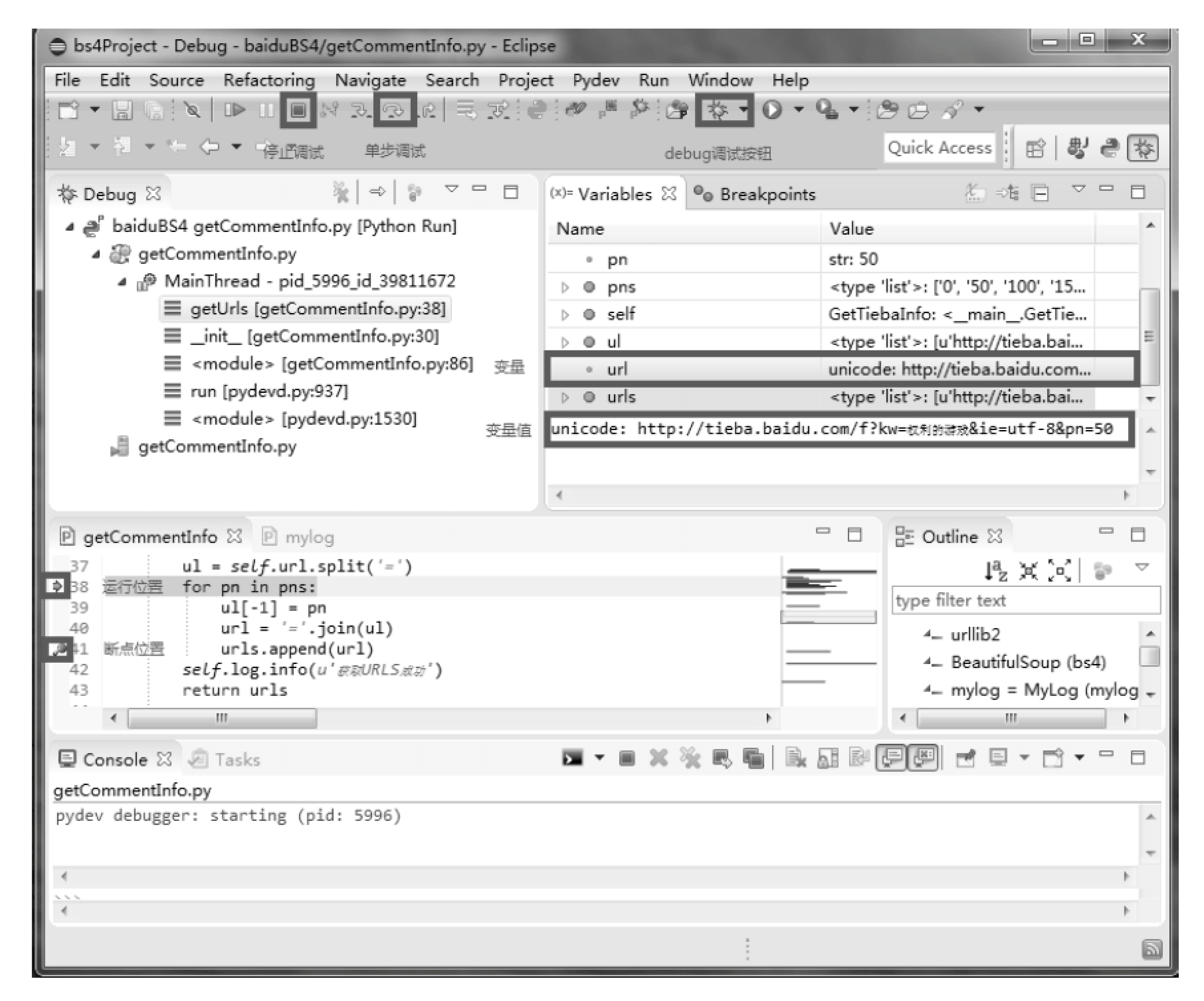


图 6-41　Eclipse 调试

配合 log 模块，即使程序出现什么问题也可以很容易地找到出错的位置，方便修改。

6.4 bs4 爬虫实战二：获取双色球中奖信息

在国内，唯一能合法暴富的方法似乎只有彩票中奖了。虽然人人都知道中奖的机率很低，但希望总是存在的。中奖的号码虽然无法直接推算出来，但根据概率计算将中奖的概率稍微调大点那还是可能的（据说所有赌场都有这样一条潜规则，不欢迎数学家进入赌场，就是为了防止客人计算概率。电影《决胜 21 点》就是根据真实事件改编，而历史上最出名的因概率计算被赌场禁止入场的人是日本的山本五十六）。在进行概率计算前要做的就是收集数据，好在中国福利彩票并不禁止收集数据进行概率计算。如何计算概率不是本章的内容，本章只负责将数据收集后存入到数据库中。

6.4.1　目标分析

在中彩网中打开双色球的往期中奖信息页面（这里使用的是 Chrome 浏览器，其他的浏览器可能会稍有区别）。网址为 http://www.zhcw.com/ssq/kaijiangshuju/index.shtml?type=0，如图 6-42 所示。



图 6-42 双色球往期中奖

在中奖信息表格上鼠标右击，选择弹出菜单中的“查看框架的源代码”。发现这个框架的数据来源于 kaijiang.zhcw.com/zhcw/html/ssq/list_1..html，如图 6-43 所示。



图 6-43 框架源代码

然后再到网页中单击“下一页”的链接，再次查看框架源代码，新的框架数据来源是 kaijiang.zhcw.com/zhcw/html/ssq/list_2.html。大致明白 URL 的变化规律了，再回头测试一下 kaijiang.zhcw.com/zhcw/html/ssq/list_1.html 是否正常，返回数据正常，如图 6-44 所示。

kaijiang.zhcw.com/zhcw/html/ssq/list_1.html

开奖日期	期号	中奖号码	销售额(元)	中奖注数		详细
				一等奖	二等奖	
2016-08-16	2016095	01 05 09 12 18 32 12	297,413,702	7 (浙 闽 粤 深圳..)	73	
2016-08-14	2016094	06 07 10 12 18 31 10	324,311,584	4 (皖 粤 深圳 陕)	83	
2016-08-11	2016093	06 09 15 17 25 27 09	298,416,216	18 (京 津 晋 沪..)	131	
2016-08-09	2016092	02 13 15 23 24 29 06	299,992,386	19 (京 蒙 辽 沪..)	97	
2016-08-07	2016091	04 08 14 22 23 28 07	329,710,234	17 (京 冀 晋 沪..)	226	
2016-08-04	2016090	02 13 17 20 21 26 07	294,008,068	23 (辽 苏 浙 皖..)	164	
2016-08-02	2016089	01 03 14 30 31 32 08	288,232,376	10 (冀 辽 浙 粤..)	48	
2016-07-31	2016088	03 14 16 18 25 33 15	320,354,148	5 (辽 苏 豫 陕)	79	
2016-07-28	2016087	02 03 10 11 14 21 12	288,396,896	3 (冀 鲁 琼)	84	

图 6-44　表格来源网页

再来看看如何获取表格中的数据。任选一个双色球往期中奖号码的框架查看源代码，这里就选末页 kaijiang.zhcw.com/zhcw/html/ssq/list_100.htm，如图 6-45 所示。



```
view-source:kaijiang.zhcw.com/zhcw/html/ssq/list_100.html

<tr>
  <td align="center">2003-04-10</td>
  <td align="center">2003014</td>
  <td align="center" style="padding-left:10px;">

  <em class="rr">03</em>

  <em class="rr">05</em>

  <em class="rr">07</em>

  <em class="rr">08</em>

  <em class="rr">21</em>

  <em class="rr">31</em>

  <em>02</em></td>
  <td><strong>12,476,130</strong></td>
  <td align="left" style="color:#999;"><strong>0</strong>

  </td>
  <td align="center"><strong class="rc">0</strong></td>
  <td align="center">
  <a href="http://www.zhcw.com/ssq/kjgg/" target="_blank"><img src="http://i
width="16" height="16" align="absmiddle" title="详细信息"/></a>
  <a href="http://www.zhcw.com/video/kaijiangshipin/" target="_blank"><img
src="http://images.zhcw.com/zhcw2010/kaijiang/zhcw/ssqpd_43.jpg" width="16" height="
  </td>
</tr>


<tr>
  <td align="center">2003-04-06</td>
  <td align="center">2003013</td>
```

图 6-45　分析数据

在做爬虫时，遇到这种表格形式的数据，那就最方便了。因为它们都有固定的标签，可

以很方便地获取数据。从图 6-45 中可以看出，表格中每一行的数据都包含在一对<tr>标签内。所以，在写爬虫时只需要先将<tr>标签挑选出来，然后再到其中过滤数据就可以了。

6.4.2 项目实施

【示例 6-5】还是打开 Eclipse，按照上节创建项目、文件的流程进行。首先创建项目 winningNumBS4，在项目中创建 PyDev Module 文件 getWinningNum.py，再把上节 baiduBS4 项目中使用过的 mylog.py 复制到 winningNumBS4 项目中，以备后期调用。项目的主文件 getWinningNum.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月7日

@author: hstking hstking@hotmail.com
'''


import re
from bs4 import BeautifulSoup
import urllib2
from mylog import MyLog as mylog


class DoubleColorBallItem(object):
    date = None  #开奖日期
    order = None  #当年的顺序
    red1 = None  #第一个红球号码
    red2 = None  #第二个红球的号码
    red3 = None  #第三个红球号码
    red4 = None  #第四个红球号码
    red5 = None  #第五个红球号码
    red6 = None  #第六个红球号码
    blue = None  #篮球号码
    money = None  #彩池金额
    firstPrize = None  #一等奖中奖人数
    secondPrize = None  #二等奖中奖人数

class GetDoubleColorBallNumber(object):
    '''这个类用于获取双色球中奖号码， 返回一个 txt 文件
    '''
    def __init__(self):
```

```
        self.urls = []
        self.log = mylog()
        self.getUrls()
        self.items = self.spider(self.urls)
        self.pipelines(self.items)


    def getUrls(self):
        '''获取数据来源网页
        '''
        URL = r'http://kaijiang.zhcw.com/zhcw/html/ssq/list_1.html'
        htmlContent = self.getResponseContent(URL)
        soup = BeautifulSoup(htmlContent, 'lxml')
        tag = soup.find_all(re.compile('p'))[-1]
        pages = tag.strong.get_text()
        for i in xrange(1, int(pages)+1):
            url = r'http://kaijiang.zhcw.com/zhcw/html/ssq/list_' + str(i) + '.html'
            self.urls.append(url)
            self.log.info(u'添加 URL:%s 到 URLS \r\n' %url)

    def getResponseContent(self, url):
        '''这里单独使用一个函数返回页面返回值，是为了后期方便地加入 proxy 和 headers 等
        '''
        try:
            response = urllib2.urlopen(url.encode('utf8'))
        except:
            self.log.error(u'Python 返回 URL:%s  数据失败  \r\n' %url)
        else:
            self.log.info(u'Python 返回 URUL:%s  数据成功 \r\n' %url)
            return response.read()


    def spider(self,urls):
        '''这个函数的作用是从获取的数据中过滤得到中奖信息
        '''
        items = []
        for url in urls:
            htmlContent = self.getResponseContent(url)
            soup = BeautifulSoup(htmlContent, 'lxml')
            tags = soup.find_all('tr', attrs={})
            for tag in tags:
                if tag.find('em'):
```

```
                item = DoubleColorBallItem()
                tagTd = tag.find_all('td')
                item.date = tagTd[0].get_text()
                item.order = tagTd[1].get_text()
                tagEm = tagTd[2].find_all('em')
                item.red1 = tagEm[0].get_text()
                item.red2 = tagEm[1].get_text()
                item.red3 = tagEm[2].get_text()
                item.red4 = tagEm[3].get_text()
                item.red5 = tagEm[4].get_text()
                item.red6 = tagEm[5].get_text()
                item.blue = tagEm[6].get_text()
                item.money = tagTd[3].find('strong').get_text()
                item.firstPrize = tagTd[4].find('strong').get_text()
                item.secondPrize = tagTd[5].find('strong').get_text()
                items.append(item)
                self.log.info(u'获取日期为:%s 的数据成功' %(item.date))
        return items

    def pipelines(self,items):
        fileName = u'双色球.txt'.encode('GBK')
        with open(fileName, 'w') as fp:
            for item in items:
                fp.write('%s %s \t %s %s %s %s %s %s  %s \t %s \t %s %s \n'
%(item.date,item.order,item.red1,item.red2,item.red3,item.red4,item.red5,item.red6,item.blue,item.money,item.firstPrize,item.secondPrize))
                self.log.info(u'将日期为:%s 的数据存入"%s"...' %(item.date, fileName.decode('GBK')))


if __name__ == '__main__':
    GDCBN = GetDoubleColorBallNumber()
```

项目中 mylog.py 的内容如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#Author  :hstking
#E-mail  :hstking@hotmail.com
#Ctime   :2015/09/15
#Mtime   :
#Version :

```

```
import logging
import getpass
import sys


#### 定义 MyLog 类
class MyLog(object):
#### 类 MyLog 的构造函数
    def __init__(self):
        self.user = getpass.getuser()
        self.logger = logging.getLogger(self.user)
        self.logger.setLevel(logging.DEBUG)

####  日志文件名
        self.logFile = sys.argv[0][0:-3] + '.log'
        self.formatter = logging.Formatter('%(asctime)-12s %(levelname)-8s %(name)-10s %(message)-12s\r\n')

####  日志显示到屏幕上并输出到日志文件内
        self.logHand = logging.FileHandler(self.logFile, encoding='utf8')
        self.logHand.setFormatter(self.formatter)
        self.logHand.setLevel(logging.DEBUG)

        self.logHandSt = logging.StreamHandler()
        self.logHandSt.setFormatter(self.formatter)
        self.logHandSt.setLevel(logging.DEBUG)

        self.logger.addHandler(self.logHand)
        self.logger.addHandler(self.logHandSt)

####  日志的 5 个级别对应以下的 5 个函数
    def debug(self,msg):
        self.logger.debug(msg)

    def info(self,msg):
        self.logger.info(msg)

    def warn(self,msg):
        self.logger.warn(msg)

    def error(self,msg):
        self.logger.error(msg)

```

```
    def critical(self,msg):
        self.logger.critical(msg)

if __name__ == '__main__':
    mylog = MyLog()
    mylog.debug(u"I'm debug 测试中文")
    mylog.info("I'm info")
    mylog.warn("I'm warn")
    mylog.error(u"I'm error 测试中文")
    mylog.critical("I'm critical")
```

鼠标左键选取项目文件 getWinningNum.py，然后左键单击 Eclipse 的运行图标并复制，最终得到结果如图 6-46 所示。

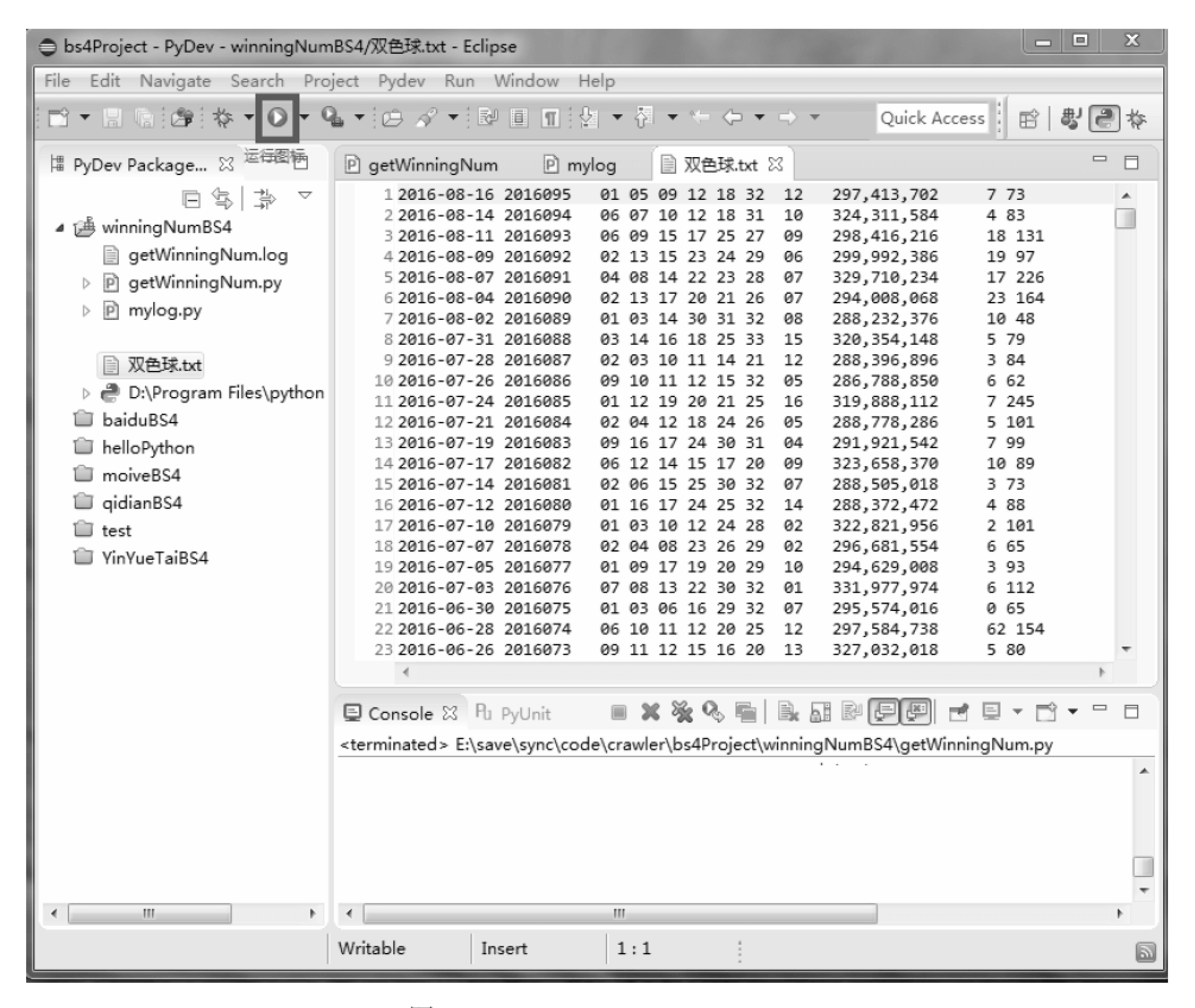


图 6-46　run getWinningNum.py

很顺利地将爬取的结果保存到了 txt 文件中。winningNumBS4 项目暂时告一段落，下节继续挖掘。

6.4.3　保存结果到 Excel

在 Windows 下，最常见的数据保存工具是 Excel。下面尝试将结果保存到 Excel 中去。

在 Python 的标准库中，并没有直接操作 Excel 的模块。好在还有永远不会让人失望的第三方库。百度一下最流行的操作 Excel 的 Python 库就是 xlrd 和 xlwt 了。其中 xlrd 负责从 Excel 中读取数据，而 xlwt 则是将数据写入到 Excel 中去。这里我们使用 xlwt 模块。

从第三方库中安装 xlwt 模块很简单，一条命令足矣。执行命令：

```
pip install xlwt
```

执行结果如图 6-47 所示。

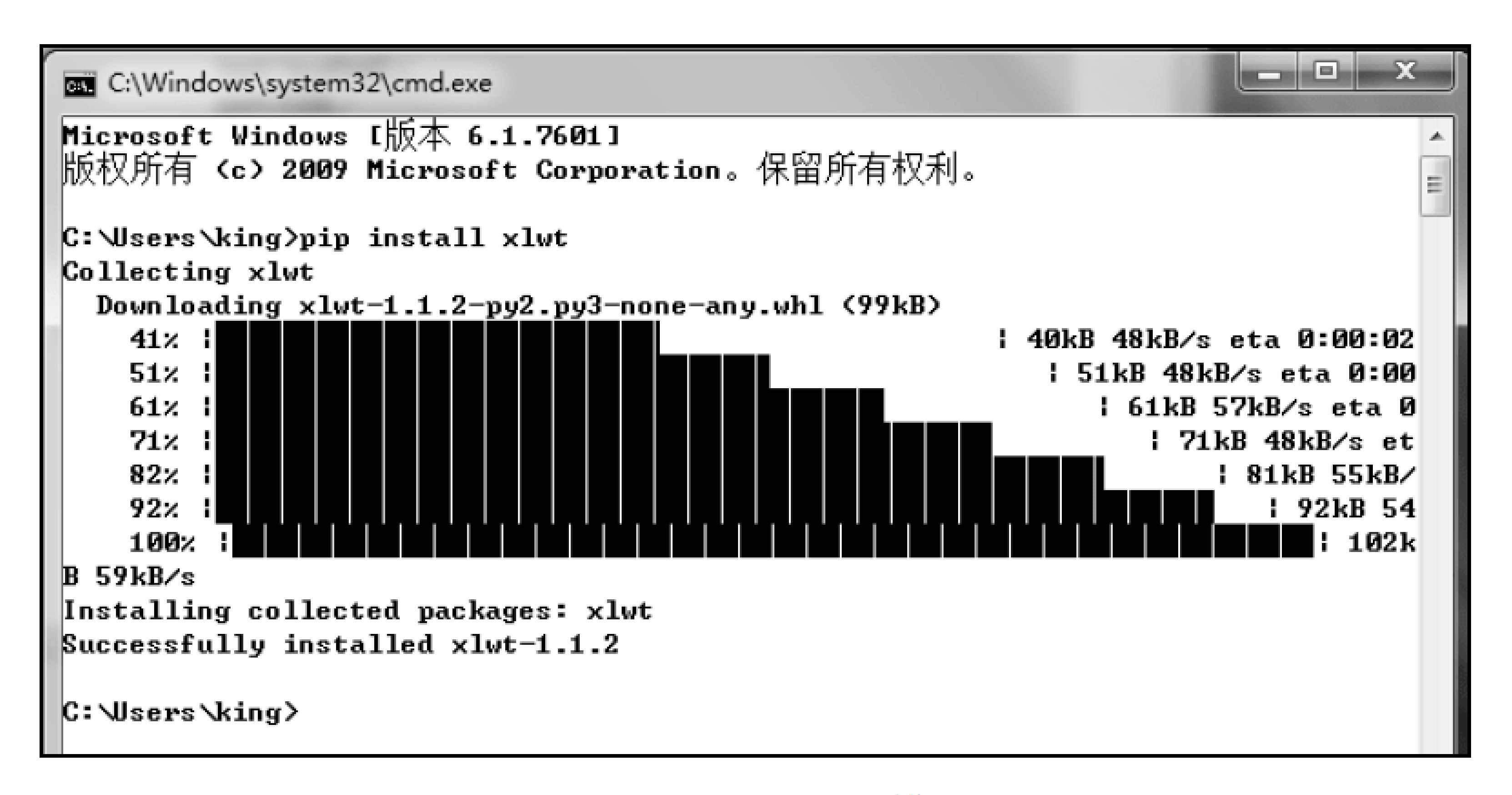


图 6-47 安装 xlwt 模块

至此 xlwt 模块已安装完毕。xlwt 模块使用很简单。

【示例 6-6】先写一个简单的 Python 程序来测试一下。在 Eclipse 中创建一个 test 项目，并在项目中创建一个名为 excelWrite.py 的 PyDev Module 文件。excelWrite.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016 年 8 月 17 日

@author: hstking hstking@hotmail.com
'''

import xlwt

if __name__ == '__main__':
    book = xlwt.Workbook(encoding='utf8', style_compression=0)
    sheet = book.add_sheet('dede')
    sheet.write(0, 0, 'hstking')
    sheet.write(1, 1, u'中文测试'.encode('utf8'))
    book.save('d:\\1.xls')
```

很简单的一个程序。首先使用 xlwt.Workbook 函数创建一个工作簿，如果有中文最好是

输入中文的字符编码。然后在工作簿里创建一个表，再就是往表里填入数据了。逻辑简单，结构也很简单。最后就是保存工作簿到文件中去。

单击 Eclipse 的运行图标，得到的结果如图 6-48 所示。

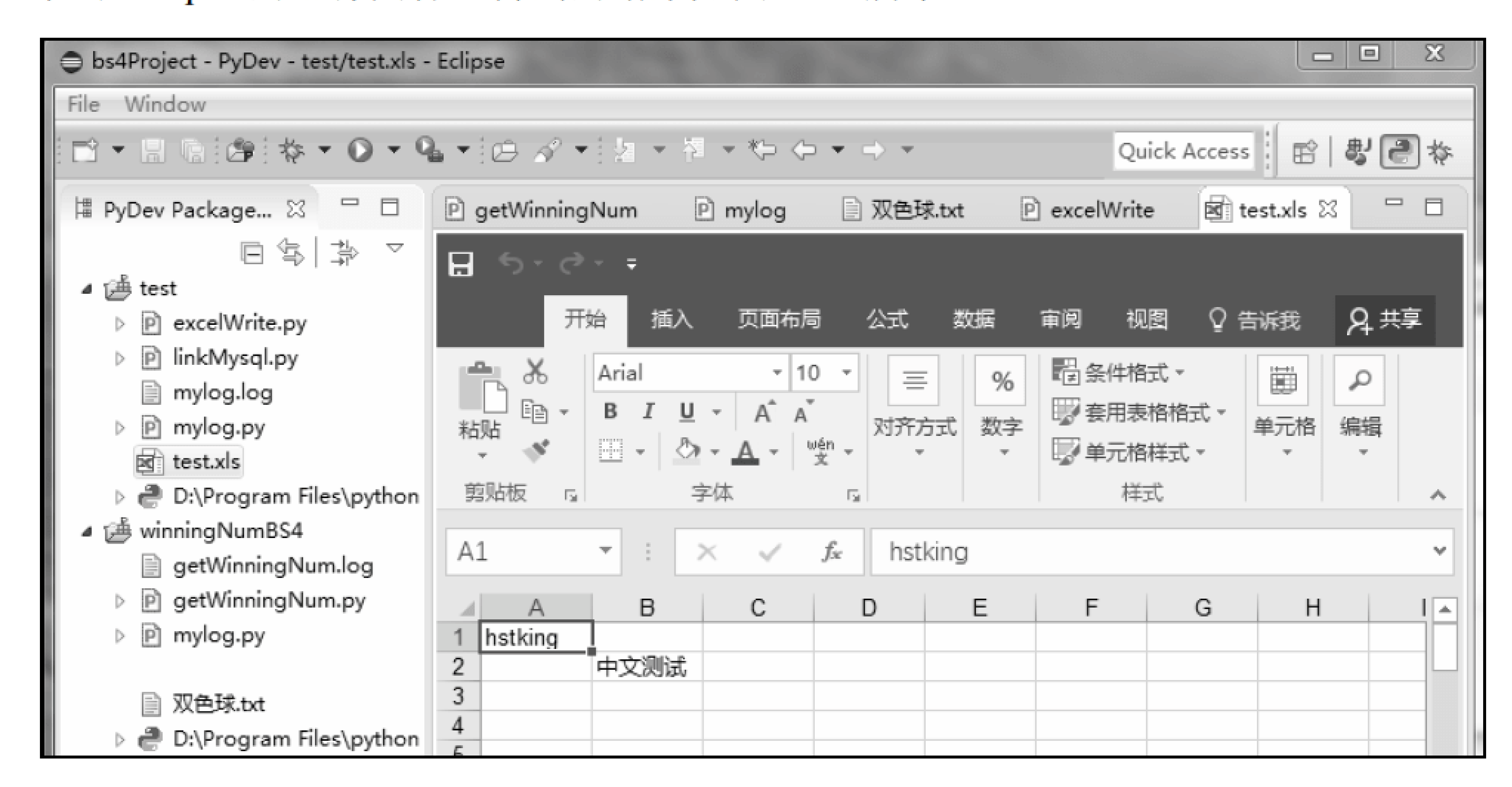


图 6-48　测试 xlwt 模块

根据 getWinningNum.log 的实际情况（也就是程序输出数据的形式），将 excelWrite.py 稍做变化就可以用了。回到 winningNumBS4 项目下，创建一个新的 PyDev Module 文件 save2excel.py、 save2excel.py 和 getWinningNum.py 是在同一目录下的（这点很重要，如果不在同一目录下，getWinningNum.py 想把 save2excel.py 当模块使用会费很多功夫）。

【示例 6-7】Sava2excel.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月17日

@author: hstking hstking@hotmail.com
'''

import xlwt


class SavaBallDate(object):
    def __init__(self, items):
        self.items = items
        self.run(self.items)

    def run(self,items):
```

```
        fileName = u'双色球.xls'.encode('GBK')
        book = xlwt.Workbook(encoding='utf8')
        sheet=book.add_sheet('ball', cell_overwrite_ok=True)
        sheet.write(0, 0, u'开奖日期'.encode('utf8'))
        sheet.write(0, 1, u'期号'.encode('utf8'))
        sheet.write(0, 2, u'红 1'.encode('utf8'))
        sheet.write(0, 3, u'红 2'.encode('utf8'))
        sheet.write(0, 4, u'红 3'.encode('utf8'))
        sheet.write(0, 5, u'红 4'.encode('utf8'))
        sheet.write(0, 6, u'红 5'.encode('utf8'))
        sheet.write(0, 7, u'红 6'.encode('utf8'))
        sheet.write(0, 8, u'蓝'.encode('utf8'))
        sheet.write(0, 9, u'销售金额'.encode('utf8'))
        sheet.write(0, 10, u'一等奖'.encode('utf8'))
        sheet.write(0, 11, u'二等奖'.encode('utf8'))
        i = 1
        while i <= len(items):
            item = items[i-1]
            sheet.write(i, 0, item.date)
            sheet.write(i, 1, item.order)
            sheet.write(i, 2, item.red1)
            sheet.write(i, 3, item.red2)
            sheet.write(i, 4, item.red3)
            sheet.write(i, 5, item.red4)
            sheet.write(i, 6, item.red5)
            sheet.write(i, 7, item.red6)
            sheet.write(i, 8, item.blue)
            sheet.write(i, 9, item.money)
            sheet.write(i, 10, item.firstPrize)
            sheet.write(i, 11, item.secondPrize)
            i += 1
        book.save(fileName)



if __name__ == '__main__':
    pass
```

【示例 6-8】将原来的 getWinningNum.py 稍做修改，修改后的 getWinningNum.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
```

```
Created on 2016年8月7日

@author: hstking hstking@hotmail.com
'''


import re
from bs4 import BeautifulSoup
import urllib2
from mylog import MyLog as mylog
from save2excel import SavaBallDate


class DoubleColorBallItem(object):
    date = None
    order = None
    red1 = None
    red2 = None
    red3 = None
    red4 = None
    red5 = None
    red6 = None
    blue = None
    money = None
    firstPrize = None
    secondPrize = None

class GetDoubleColorBallNumber(object):
    '''这个类用于获取双色球中奖号码, 返回一个txt文件
    '''
    def __init__(self):
        self.urls = []
        self.log = mylog()
        self.getUrls()
        self.items = self.spider(self.urls)
        self.pipelines(self.items)
        self.log.info('beging save data to excel \r\n')
        SavaBallDate(self.items)
        self.log.info('save data to excel end ...\r\n')


    def getUrls(self):
        '''获取数据来源网页
        '''
        URL = r'http://kaijiang.zhcw.com/zhcw/html/ssq/list_1.html'
        htmlContent = self.getResponseContent(URL)
        soup = BeautifulSoup(htmlContent, 'lxml')
```

```
        tag = soup.find_all(re.compile('p'))[-1]
        pages = tag.strong.get_text()
        for i in xrange(1, int(pages)+1):
            url = r'http://kaijiang.zhcw.com/zhcw/html/ssq/list_' + str(i) + '.html'
            self.urls.append(url)
            self.log.info(u'添加 URL:%s 到 URLS \r\n' %url)

    def getResponseContent(self, url):
        '''这里单独使用一个函数返回页面返回值，是为了后期方便地加入 proxy 和 headers 等
        '''
        try:
            response = urllib2.urlopen(url.encode('utf8'))
        except:
            self.log.error(u'Python 返回 URL:%s  数据失败  \r\n' %url)
        else:
            self.log.info(u'Python 返回 URUL:%s  数据成功 \r\n' %url)
            return response.read()


    def spider(self,urls):
        '''这个函数的作用是从获取的数据中过滤得到中奖信息
        '''
        items = []
        for url in urls:
            htmlContent = self.getResponseContent(url)
            soup = BeautifulSoup(htmlContent, 'lxml')
            tags = soup.find_all('tr', attrs={})
            for tag in tags:
                if tag.find('em'):
                    item = DoubleColorBallItem()
                    tagTd = tag.find_all('td')
                    item.date = tagTd[0].get_text()
                    item.order = tagTd[1].get_text()
                    tagEm = tagTd[2].find_all('em')
                    item.red1 = tagEm[0].get_text()
                    item.red2 = tagEm[1].get_text()
                    item.red3 = tagEm[2].get_text()
                    item.red4 = tagEm[3].get_text()
                    item.red5 = tagEm[4].get_text()
                    item.red6 = tagEm[5].get_text()
                    item.blue = tagEm[6].get_text()
                    item.money = tagTd[3].find('strong').get_text()
                    item.firstPrize = tagTd[4].find('strong').get_text()
                    item.secondPrize = tagTd[5].find('strong').get_text()
                    items.append(item)
                    self.log.info(u'获取日期为:%s 的数据成功' %(item.date))
```

```
        return items

    def pipelines(self,items):
        fileName = u'双色球.txt'.encode('GBK')
        with open(fileName, 'w') as fp:
            for item in items:
                fp.write('%s %s \t %s %s %s %s %s %s  %s \t %s \t %s %s \n'
%(item.date,item.order,item.red1,item.red2,item.red3,item.red4,item.red5,item.red6,item.blue,item.money,item.firstPrize,item.secondPrize))
                self.log.info(u'将日期为:%s 的数据存入"%s"...' %(item.date, fileName.decode('GBK')))


if __name__ == '__main__':
    GDCBN = GetDoubleColorBallNumber()
```

实际上就是导入了 save2excel 模块后再在__init__函数中添加了 3 行，让 save2excel 处理了一下爬取的数据而已。

单击 Eclipse 图标栏的运行图标，最终得到的结果如图 6-49 所示。

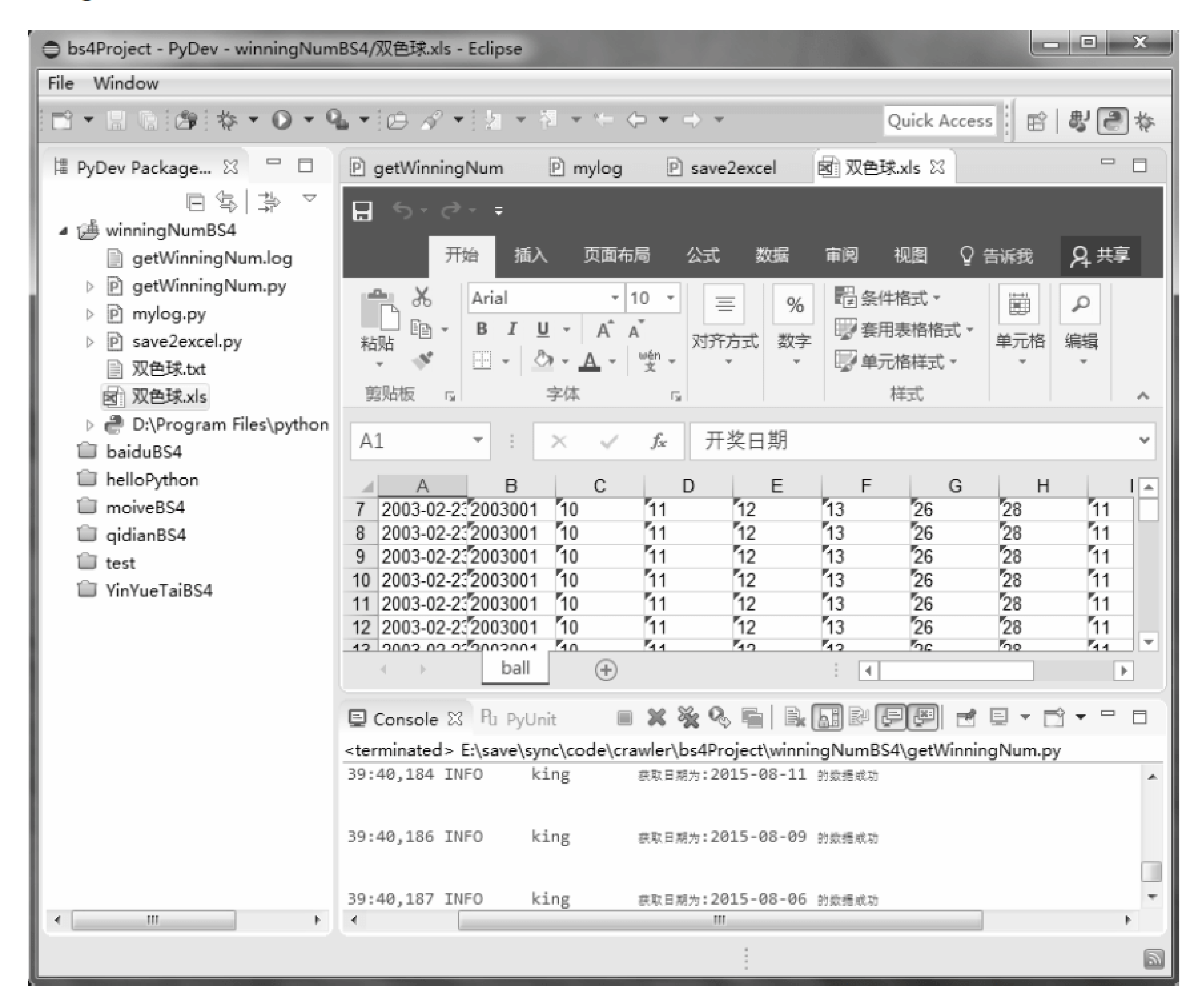


图 6-49　保存结果到 excel

至此，winningNumBS4 项目已全部完成。这里要注意的是，不要频繁地跑这个爬虫，在一定的时间内多次爬取会引起网站反爬虫人员的注意。得到数据就够了，没必要耗费网站的网络资源。

6.4.4　代码分析

winningNumBS4 项目中只有 3 个 Python 文件，分别是 getWinningNum.py、save2excel.py 和 mylog.py。其中 mylog.py 和上个项目中的 mylog.py 是一样的（实际上所有的项目中使用的都是同一个 mylog.py），这个可以略过。主要是看主文件 getWinningNum.py 和 save2excel.py。

先看 getWinningNum.py 文件。第 10~14 行是导入程序所需的模块。其中包含了 Python 标准模块和自定义模块。顺便演示了导入模块的几种方式。

第 17~29 行定义了一个类。这个类包含了获取数据的所有项。这种写法借鉴了 Scrapy 的 items 的写法。第 31~105 行定义的模块负责获取数据和保存数据。其中 34~42 行是类的解析函数（Python 是没有解析函数这个概念的，但__init__函数所起的作用基本上就是解析函数了）。当类实例化时就自动运行__init__函数。

第 45~56 行的 getUrls 函数作用就是从网站中获取所有需要爬取的网页的 URL，然后将这些 url 加入到 urls 列表中去，以备后面的函数调用。第 58~67 行 getResponseConent 函数作用就是从一个 url 中获取数据。这个函数可以用一句话代替，之所以写成函数是为了加入 headers 和 proxy 方便。

第 70~97 行是使用爬虫爬取数据，然后将爬取的数据保存到 items 列表中供后面的函数调用。第 99~105 行 pipelines 函数将 items 列表中的数据写入 txt 文件中。那么将数据写入 Excel 是哪个函数呢？是在第 41 行，从__init__函数中调用了 save2excel 模块中的 SaveBallDate 函数。

最后的第 108~109 行是主函数，它将实例化 GetDoubleColorBallNumber 类。使 GetDoubleColorBallNumber 类的__init__函数自动运行。

至于 save2excel.py 就比较简单了。第 9 行导入了 xlwt 模块。第 13~14 行初始化了类变量，调用类函数。第 17~49 行就是很简单地遍历 items 列表，然后将列表中的数据保存到 Excel 中去。稍微要注意的是第 18 行，因为是 Windows 系统，默认使用 GBK 字符编码。如果不希望中文文件名出现乱码，那就将含有中文的文件名转换成 GBK 编码。

6.5　bs4 爬虫实战三：获取起点小说信息

现在娱乐消费好贵。最便宜最方便的娱乐消费莫过于在网上看小说了（网游还需要充值，看电影一次至少两小时），看小说就绕不开原创中文小说网站——起点中文。笔者看小说不喜欢那种看一章等一章的看法，说不定一不小心小说就夭折了，还是直接看那些完本小

说比较爽快。本节将使用 bs4 爬虫获取起点中文网所有的完本小说信息，并将其保存到 MySQL 中去。

6.5.1 目标分析

打开起点中文网，搜索所有的完本小说。将打开网页 http://all.qidian.com/Book/BookStore.aspx?ChannelId=-1&SubCategoryId=-1&Tag=all&Size=-1&Action=5&OrderId=6&P=all&PageIndex=1&update=-1&Vip=-1&Boutique=-1&SignStatus=-1，在网页中将写作进度改成“已经完本”，其他的选项都选择“不限”或者“默认”，如图 6-50 所示。



图 6-50　选择小说类型

再来看看 URL，这个参数似乎太多了点。仔细观察一下，这个参数可能跟小说的类型选择有关系。每次削减一个参数试试页面是否有变化，最后发现 URL 为 http://all.qidian.com/Book/BookStore.aspx?&Tag=all&Action=5&OrderId=6&P=all&PageIndex=1 时页面是不变的，可以正常使用。

下面将这个 URL 中的 PageIndex=1 改成 PageIndex=2 试试。浏览器中打开 http://all.qidian.com/Book/BookStore.aspx?&Tag=all&Action=5&OrderId=6&P=all&PageIndex=2，页面也没什么问题，返回也是正常的，那么所需爬取页面的 urls 列表规律就明了了，就是修改 PageIndex 参数而已。

总共需要爬取多少页呢？查看页面中表格下方的页面选择位置，共有 144 页，如图 6-51 所示。

197	[灵异/恐怖惊悚]	红印 梦	2571	爱独不毒	16-08-12 11:35
198	[科幻/进化变异]	七星梦魇 第十五章 多重人格分裂症	50283	离枯	16-08-19 17:15
199	[科幻/时空穿梭]	穿梭万界拯救宇宙联邦 完本感言	114461	桃李成荫	16-08-10 09:34
200	[都市/异术超能]	校园正太高手 完本感言! [VIP]	3513327	日月当歌	16-08-17 09:55

首页 | 1 2 3 4 5 6 7 8 9 10 11 | 末页 | 到 页 GO　2/144 页 共 14380 篇

图 6-51　总页数

在网页的任意空白处右击，选择弹出菜单中的“查看网页源代码”，找到表格的开头部

分，如图 6-52 所示。

```
<!--[if !IE]> 结果列表 开始 <![endif]-->
<div class="sw2"><div class='swz'>101</div><div class="swa">[<a href="http://all.qidian.com/Book/BookStore.aspx?ChannelId=4"
class="hui2">都市</a>/<a href="http://all.qidian.com/Book/BookStore.aspx?ChannelId=4&SubCategoryId=74"   class="hui2">异术超能</a>]</div><div
class="swb"><span class="swbt"><a href="http://www.qidian.com/Book/1003682218.aspx" target="_blank">龙斗魔王</a> </span><a
href="http://read.qidian.com/BookReader/8UUQocy4n5POVUeyz9PqUQ2,jbpppkBNR9VOBDFlr9quQA2.aspx" target="_blank" class="hui2">第三十五章 尾声</a>
</div>      <div class="swc">29618</div><div class="swd"><a href="http://me.qidian.com/authorIndex.aspx?id=7838379" target="_blank"
class="black">公子猪</a></div>         <div class="swe">16-08-16 18:34</div>        </div><div class="sw1"><div class='swz'>102</div><div
class="swa">[<a href="http://all.qidian.com/Book/BookStore.aspx?ChannelId=21" class="hui2">玄幻</a>/<a
href="http://all.qidian.com/Book/BookStore.aspx?ChannelId=21&SubCategoryId=8"   class="hui2">东方玄幻</a>]</div><div class="swb"><span
class="swbt"><a href="http://www.qidian.com/Book/3682155.aspx" target="_blank">符道至圣</a> </span><a rel="nofollow"
href="http://vipreader.qidian.com/BookReader/vip,3682155,316937899.aspx" target="_blank" class="hui2">第三百八十四章 天外天（大结局）</a> <span
class="gt">[VIP]</span> </div>      <div class="swc">774494</div><div class="swd"><a href="http://me.qidian.com/authorIndex.aspx?id=3887914"
target="_blank" class="black">风靡网络</a></div>         <div class="swe">16-08-15 12:43</div>        </div><div class="sw2"><div
class='swz'>103</div><div class="swa">[<a href="http://all.qidian.com/Book/BookStore.aspx?ChannelId=12" class="hui2">二次元</a>/<a
href="http://all.qidian.com/Book/BookStore.aspx?ChannelId=12&SubCategoryId=281"   class="hui2">衍生同人</a>]</div><div class="swb"><span
class="swbt"><a href="http://www.qidian.com/Book/3676971.aspx" target="_blank">假面骑士之降临绝世唐门</a> </span><a
href="http://read.qidian.com/BookReader/bKeiJULyZH41,1gzrGq8Pry5Ms5iq0oQwLQ2.aspx" target="_blank" class="hui2">作品相关  新书上架</a>  </div>
<div class="swc">82090</div><div class="swd"><a href="http://me.qidian.com/authorIndex.aspx?id=5122195" target="_blank" class="black">绝世
ghos...</a></div>         <div class="swe">16-08-14 21:01</div>        </div><div class="sw1"><div class='swz'>104</div><div class="swa">[<a
href="http://all.qidian.com/Book/BookStore.aspx?ChannelId=9" class="hui2">科幻</a>/<a href="http://all.qidian.com/Book/BookStore.aspx?
ChannelId=9&SubCategoryId=253"   class="hui2">末世危机</a>]</div><div class="swb"><span class="swbt"><a
href="http://www.qidian.com/Book/1003742452.aspx" target="_blank">末世之灭尸系统</a> </span><a
```

图 6-52 表格源码

表格的所有行不是以<div class="sw2">开头，就是以<div class="sw1">开头。没问题，只怕没有规律的，不怕规律多的。已知 url 的数量和规律，也知道了寻找表格数据的方法。这个爬虫已经完成一半了，后面纯粹就是编程的问题，那就简单了。

6.5.2 项目实施

打开 Eclipse，创建项目 qidianBS4，在项目中创建 PyDev Module 文件 completeBook.py。把上节 winningNumBS4 项目中使用过的 mylog.py 复制到 qidianBS4 项目中，以备后期调用。

【示例 6-9】项目的主文件 completeBook.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月11日

@author: hstking hstking@hotmail.com
'''

from bs4 import BeautifulSoup
import urllib2
import re
import codecs
import time
from mylog import MyLog as mylog
from save2mysql import SavebooksData


class BookItem(object):
```

```
    categoryName = None  #小数类型
    middleUrl = None
    bookName = None  #小说名
    wordsNum = None  #总字数
    updateTime = None  #更新时间
    authorName = None  #作者


class GetBookName(object):
    def __init__(self):
        self.urlBase = 'http://all.qidian.com/Book/BookStore.aspx?&Tag=all&Action=5&OrderId=6&P=all&PageIndex=1'
        self.log = mylog()
        self.pages = self.getPages(self.urlBase)
        self.booksList = []
        self.spider(self.urlBase, self.pages)
        self.pipelines(self.booksList)
        self.log.info('begin save data to mysql\r\n')
        SavebooksData(self.booksList)
        self.log.info('save data to mysql end ...\r\n')


    def getPages(self,url):
        htmlContent = self.getResponseContent(url)
        soup = BeautifulSoup(htmlContent, 'lxml')
        tags = soup.find('div', attrs={'class':'storelistbottom'})
        strUrl = tags.find_all('a')[-1].get('href')
        for st in strUrl.split('&'):
            if re.search('PageIndex', st):
                pages = st.split('=')[-1]
                self.log.info(u'获取页数为:%s' %pages)
                return int(pages)


    def getResponseContent(self, url):
        try:
            response = urllib2.urlopen(url.encode('utf8'))
        except:
            self.log.error(u'Python 返回 URL:%s  数据失败' %url)
        else:
            self.log.info(u'Python 返回 URUL:%s  数据成功' %url)
            return response.read()
```

```


    def spider(self, url, pages):
        urlList = url.split('=')
        for i in xrange(1, pages + 1):
            urlList[-1] = str(i)
            newUrl = '='.join(urlList)
            htmlContent = self.getResponseContent(newUrl)
            soup = BeautifulSoup(htmlContent, 'lxml')
            tags = soup.find_all('div', attrs={'class':'sw1'}) + soup.find_all('div', attrs={'class': 'sw2'})
            for tag in tags:
                item = BookItem()
                item.categoryName = tag.find('div', attrs={'class': 'swa'}).get_text()
                item.middleUrl = tag.find('div', attrs={'class':'swb'}).span.a.get('href')
                item.bookName = tag.find('div', attrs={'class': 'swb'}).span.a.get_text()
                item.wordsNum = tag.find('div', attrs={'class':'swc'}).get_text()
                item.updateTime = tag.find('div', attrs={'class':'swe'}).get_text()
                item.authorName = tag.find('div', attrs={'class':'swd'}).a.get_text()
                self.booksList.append(item)
                self.log.info(u'获取书名为<<%s>>的数据成功' %item.bookName)


    def pipelines(self,bookList):
        bookName = u'起点完本小说.txt'.encode('GBK')
        nowTime = time.strftime('%Y-%m-%d %H:%M:%S\r\n', time.localtime())
        with codecs.open(bookName, 'w', 'utf8') as fp:
            fp.write('run time: %s' %nowTime)
            for item in self.booksList:
                fp.write('%s \t %s \t\t %s \t %s \t %s \r\n'
                         %(item.categoryName, item.bookName, item.wordsNum, item.updateTime, item.authorName))
                self.log.info(u'将书名为<<%s>>的数据存入"%s"...' %(item.bookName, bookName.decode('GBK')))

if __name__ == '__main__':
    GBN = GetBookName()
```

主程序部分已经完成了。

6.5.3 保存结果到 MySQL

因为最后还需要将结果保存到 MySQL 中去，所以还得添加一个自定义的模块（也就是一个自建 Python 程序）。

【示例 6-10】在 qidianBS4 项目下，completeBook.py 的同级目录中创建一个 PyDev Module 文件 save2mysql.py。save2mysql.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月17日

@author: hstking hstking@hotmail.com
'''

import MySQLdb

class SavebooksData(object):
    def __init__(self,items):
        self.host = '192.168.2.80'  #mysql服务器IP
        self.port = 3306  #mysql服务器端口
        self.user = 'crawlUSER'  #mysql服务器用户名
        self.passwd = 'crawl123'  #mysql服务器密码
        self.db = 'bs4DB'  #使用的库名

        self.run(items)

    def run(self, items):
        conn = MySQLdb.connect(host=self.host,
                               port=self.port,
                               user=self.user,
                               passwd=self.passwd,
                               db=self.db,
                               charset='utf8')
        cur = conn.cursor()
        for item in items:
            cur.execute("INSERT INTO qiDianBooks(categoryName, bookName, wordsNum, updateTime, authorName) values(%s, %s, %s, %s, %s)", (item.categoryName, item.bookName, item.wordsNum, item.updateTime, item.authorName))
        cur.close()
```

```
        conn.commit()
        conn.close()


if __name__ == '__main__':
    pass
```

至此，该项目中的所有代码已经完成了。因为要将数据保存到远程 MySQL 服务器，首先得在 MySQL 服务器上创建好数据库和表。在第 5 章 Scrapy 项目中就创建过一个 MySQL 服务器，并分别在 Windows 和 Linux 系统上安装了 Python 中支持 MySQL 的模块 MySQLdb。这里就直接使用现成的 MySQL 服务器就可以了。与 Scrapy 项目不同的是，本节存储数据到 MySQL 服务器是远程存储，Scrapy 项目的存储是本地存储。本机与 MySQL 服务器的关系如图 6-53 所示。

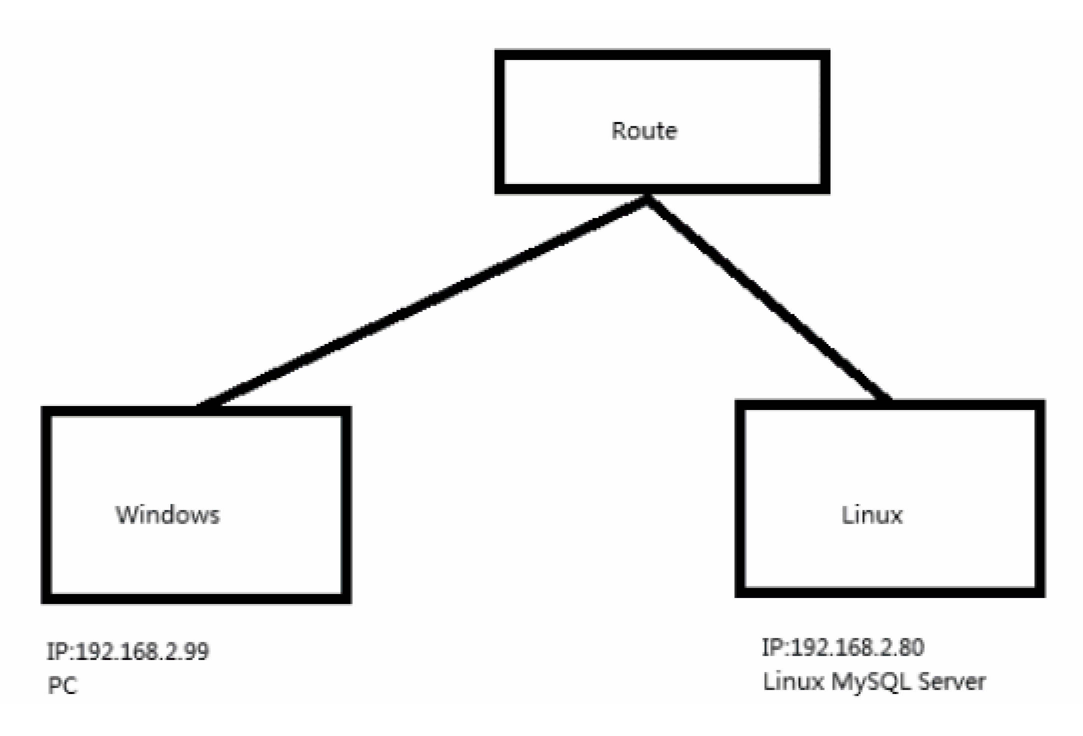


图 6-53　网络关系图

使用 putty 登录到 Linux。进入 MySQL 为 bs4 项目创建一个数据库，并为 qidianBS4 项目创建一个表。

在 Linux 系统下登录到 MySQL 后，执行命令：

```
CREATE DATABASE bs4DB CHARACTER SET 'utf8' COLLATE 'utf8_general_Ci';
use bs4DB;
CREATE TABLE qiDianBooks(
    -> id INT AUTO_INCREMENT,
    -> categoryName char(20),
    -> bookName char(20),
    -> wordsNum char(10),
    -> updateTime char(20),
    -> authorName char(20),
-> PRIMARY KEY(id) )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

创建一个名为 bs4DB 的数据库，并在数据库中建立一个 qiDianBooks 的表，所有编码都使用 utf8 编码，执行结果如图 6-54 所示。

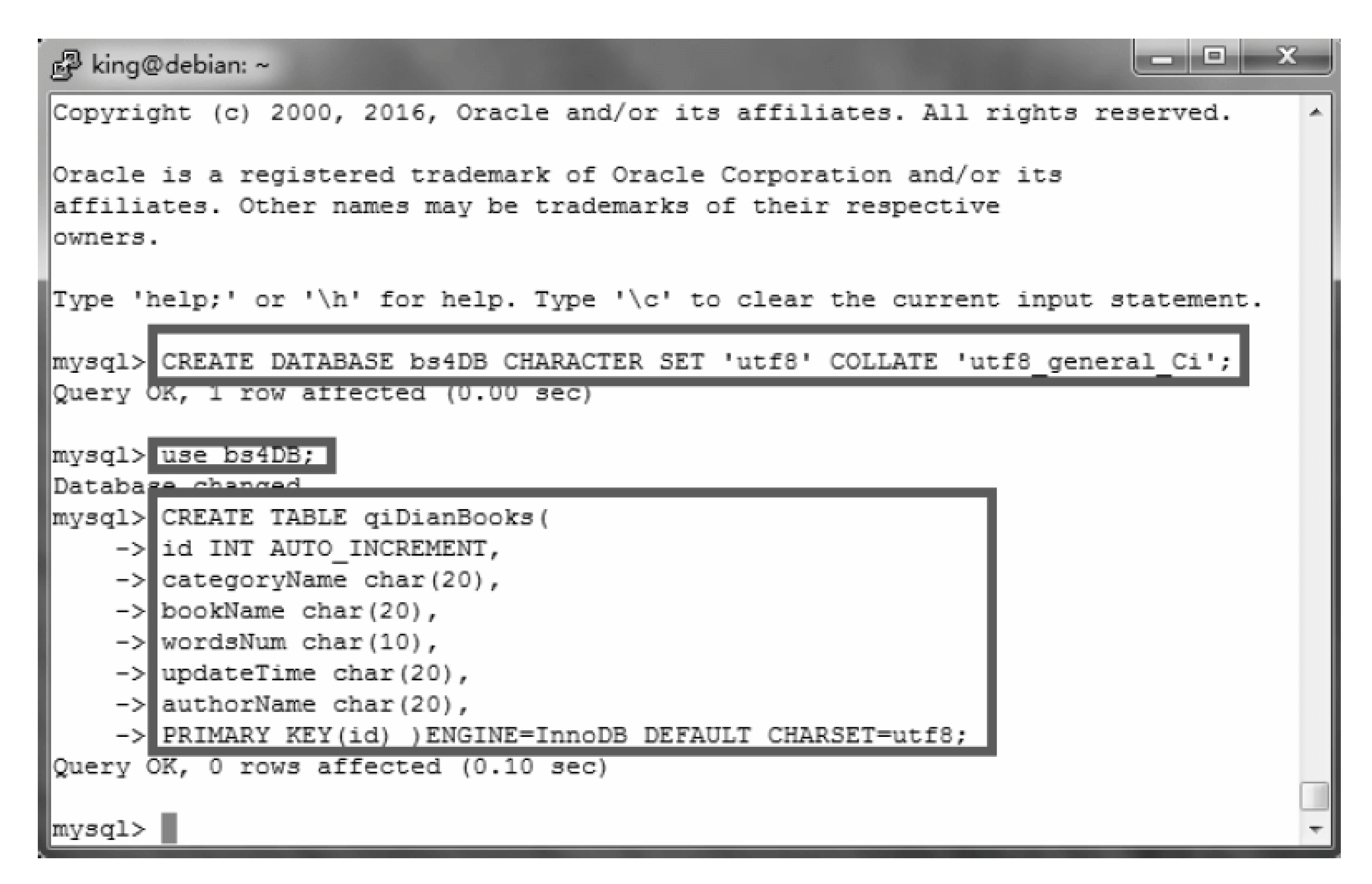


图 6-54　创建数据库和表

随后为登录用户分配权限，执行命令：

```
GRANT all privileges ON bs4DB.* to crawlUSER@all IDENTIFIED BY 'crawl123';
GRANT all privileges ON bs4DB.* to crawlUSER@localhost IDENTIFIED BY
'crawl123';
GRANT all privileges ON bs4DB.* to crawlUSER@192.168.2.99 IDENTIFIED BY
'crawl123';
```

执行结果如图 6-55 所示。

```
mysql>
mysql> GRANT all privileges ON bs4DB.* to crawlUSER@all IDENTIFIED BY 'crawl123'
;
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT all privileges ON bs4DB.* to crawlUSER@localhost IDENTIFIED BY 'cra
wl123';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT all privileges ON bs4DB.* to crawlUSER@192.168.2.99 IDENTIFIED BY '
crawl123';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

图 6-55　MySQL 用户权限

这三条命令的作用分别是：允许 crawlUSER 用户远程登录，允许 crawlUSER 用户本地登录，允许 crawlUSER 从 192.168.2.99 这个 IP 登录。

这里需要说明的是，crawlUSER 这个用户在 Scrapy 项目中已经创建过了，如果没有创建这个用户，则需要在 MySQL 中执行命令：

```
INSERT INTO mysql.user(Host,User,Password)
VALUES("%","crawlUSER",password("crawl123"));
INSERT INTO mysql.user(Host,User,Password)
VALUES("localhost","crawlUSER",password("crawl123"));
```

MySQL 服务器已经准备好了，可以运行程序了。单击 Eclipse 图标栏上的运行图标，执行该项目，得到的结果如图 6-56 所示。

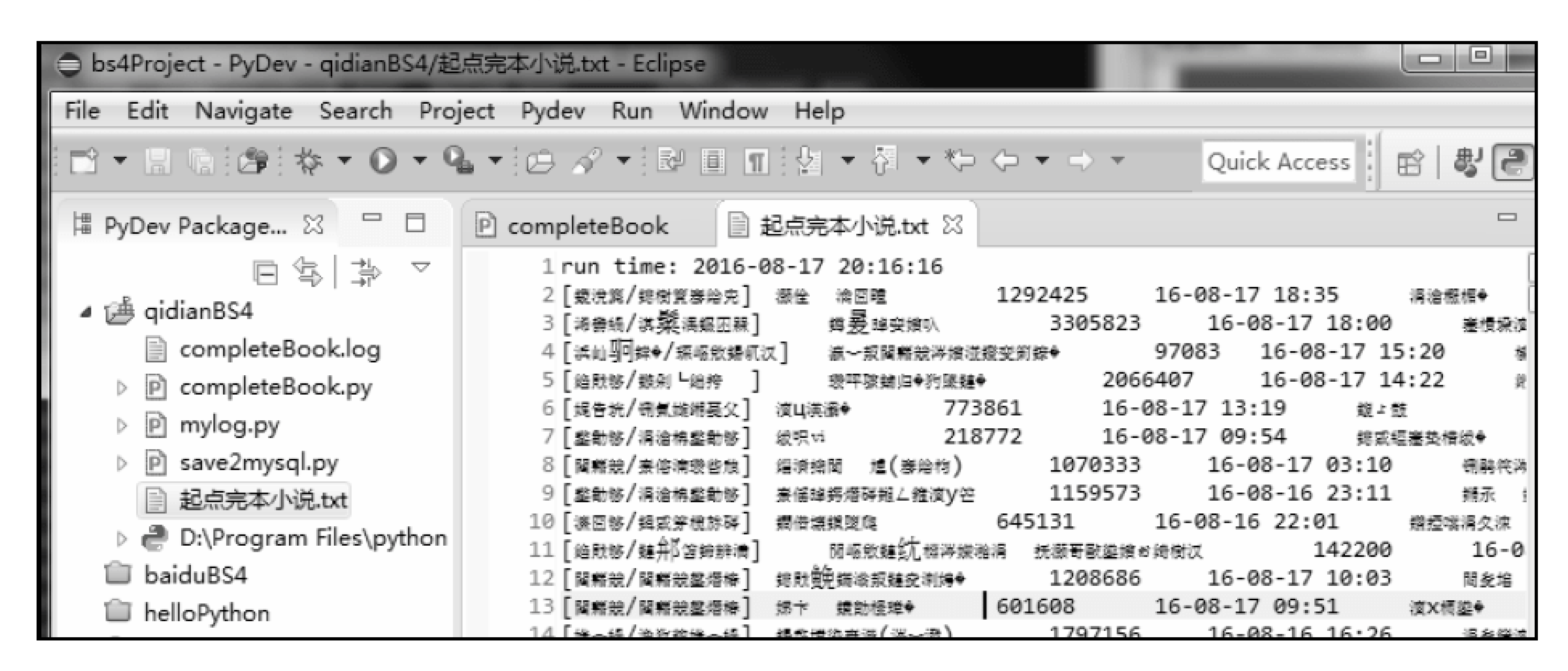


图 6-56　run completeBook.py

保存到本地的文件内容有乱码，是不是？没关系，用 Windows 自带的记事本打开就没有了，只是编码问题。再来看看保存到远程的 MySQL。使用 pytty 登录到 Linux，连接到 MySQL 服务器后，在 MySQL 里执行命令：

```
use bs4DB;
select * from qiDianBooks;
```

执行的结果如图 6-57 所示。

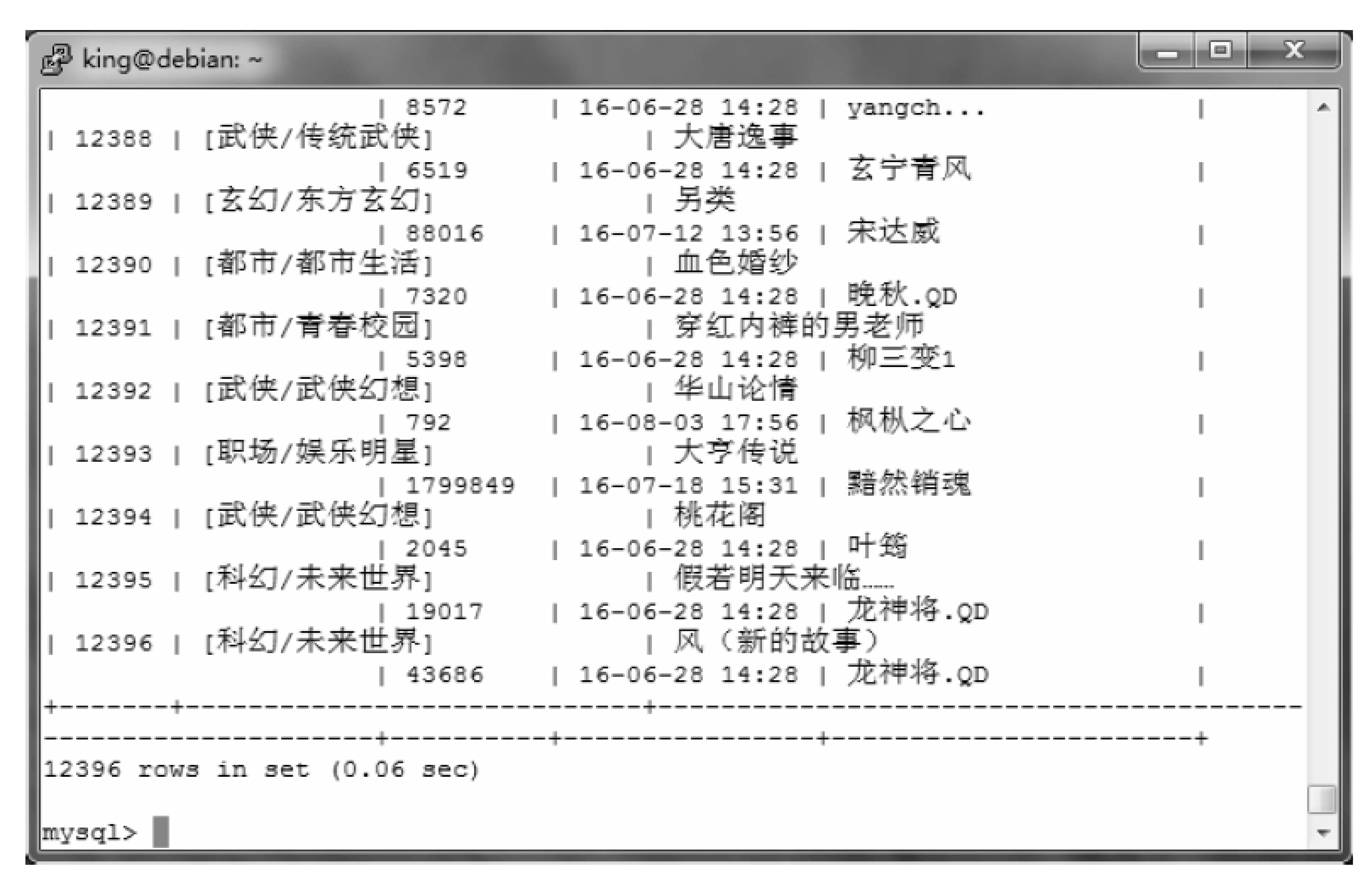


图 6-57　MySQL 保存结果

保存的结果是 12396 条，而网站上显示所有的全本小说是 14380 本，大约有 2000 本小说没有被录上。稍微修改一下程序，可以从 log 中得到哪些小说没有被爬取。笔者大致检查了一下，有一些小说记录的标签与笔者设置的标签不太一样，如果需要读取完整的列表请读者自行修改一下。另外这个程序的瑕疵在于 MySQL 保存数据时使用的都是 char 类型，如果要

追求完美，可以将日期改成 Date 类型，将字数改成 int 类型等。

6.5.4 代码分析

项目 qidianBS4 中有 3 个 Python 程序，其中 mylog.py 无须再解释了，这里需要分析的只有 completeBook.py 和 save2mysql.py。

completeBook.py 中第 9~15 行是导入需要的模块。第 18~24 行是仿照 Scrapy 建立的一个 item 类。第 27~90 行是自定义类 GetBookName，用于从起点中文网站获取全本小说的信息。

第 27~37 行是 GetBookName 类的“解析函数”。第 33 行调用了 spider 类函数将所有的数据保存到类变量 self.booksList 列表中。第 34 和 36 行分别调用了 self.pipelines 函数和 SavebookData 函数，将所爬取的数据保存到 txt 文件和 MySQL 远程服务器中。

第 40~49 行是从网站起始页面中获取了总共的页数。因为这个总页数并不是单独包含在某个标签内的，所以这里使用 re 模块将这个页数过滤出来。

第 52~59 行的 getResponseContent 函数还是用于返回从 URL 中得到的原始数据。

第 62~79 行是使用 bs4 模块从原始数据中过滤出所需要的数据，然后将数据保存到了 self.booksList 列表中去。

第 82~90 行的 pipelines 函数将 self.bookList 列表中的数据保存到 txt 文件。

第 92~93 行是__main__程序，只有一条命令，作用是实例化 GetBookName 类。

6.6 bs4 爬虫实战四：获取电影信息

这一节的内容还是跟娱乐有关，从网络上获取电影。影视网站会每天都有新的电影上架，限于页面的篇幅，每页显示的电影有限。如果想找一部心仪的电影恐怕得翻遍整个网站，当然也可以使用百度的高级搜索和网站自身的搜索，但最方便的还是自己写个爬虫，每天让它爬一次，就可以知道有什么新电影上架了。

6.6.1 目标分析

这次爬虫的目标网站是 http://dianying.2345.com/，爬虫的搜索目标仅限于今年的电影。在网站打开搜索，在年代中选择 2016，得到的结果如图 6-58 所示。



图 6-58　爬虫起始页

从图片上看，这个站点和上节的起点找书没什么区别啊？的确如此，从爬虫上看，除了爬取的规则有所不同外，这个爬虫和起点网站的爬虫区别不大。这节的重点不在于爬虫，而在于获取页面的过程，这个暂且先放下，继续爬虫过程。

在页面的下方单击“下一页”，发现 URL 变成了 http://dianying.2345.com/list/----2016---2.html。测试一下 http://dianying.2345.com/list/----2016---1.html，可以正常返回，urls 的变化规律找到了。再看看总共有多少页呢？如图 6-59 所示。



图 6-59　总页数

总页数也找到了，最后只需要找到爬虫的过滤规则就可以了。单击页面空白处，在弹出菜单中选择“查看网页源代码”选项，查看页面源代码，如图 6-60 所示。

```
<ul class="globalPicTxt pic140 clearfix">
        <li media="161388">
        <div class="pic" onmouseover="this.className='pic picHover';" onmouseout="this.className='pic';">
            <img width="140" height="195" src="http://imgwx5.2345.com/dypcimg/images/noimg.png"
loadsrc="http://imgwx5.2345.com/dypcimg/img/a/53/s161388.jpg"
onerror="javascript:this.src='http://imgwx3.2345.com/dypcimg/images/default_img110.jpg';" alt="疯狂动物城国语版">
            <i class="iStyleIcon iStyleFF"></i>            <span class="pRightBottom"><em>9.8分</em></span>
            <a class="aPlayBtn" href="http://dianying.2345.com/detail/161388.html" target="_blank" title="疯狂动物城国
ajax83="ys_dy_list_pic_161388"><i></i></a>
        </div>
        <span class="sTit"><a title="疯狂动物城国语版" target="_blank" href="http://dianying.2345.com/detail/161388.ht
ajax83="ys_dy_list_title_161388">疯狂动物城（国语版&#8230;</a></span>
        <span class="sDes">主演: <em><a href="http://dianying.2345.com/mingxing/22403/" title='金妮弗·古德温电影' 1 t
ajax83="ys_dy_list_actor_161388" >金妮弗·古德温</a></em> 杰森·贝特曼</span>
    </li>
        <li media="168580">
        <div class="pic" onmouseover="this.className='pic picHover';" onmouseout="this.className='pic';">
            <img width="140" height="195" src="http://imgwx1.2345.com/dypcimg/images/noimg.png"
loadsrc="http://imgwx2.2345.com/dypcimg/img/9/56/s168580.jpg"
onerror="javascript:this.src='http://imgwx4.2345.com/dypcimg/images/default_img110.jpg';" alt="致青春原来你还在这里">
            <i class="iStyleIcon iStyleFF"></i>            <span class="pRightBottom"><em>6.5分</em></span>
            <a class="aPlayBtn" href="http://dianying.2345.com/detail/168580.html" target="_blank" title="致青春原来你
ajax83="ys_dy_list_pic_168580"><i></i></a>
        </div>
        <span class="sTit"><a title="致青春原来你还在这里" target="_blank" href="http://dianying.2345.com/detail/16858
ajax83="ys_dy_list_title_168580">致青春: 原来你还在&#8230;</a></span>
        <span class="sDes">主演: <em><a href="http://dianying.2345.com/list/---wuyifan---.html" title='刘亦菲电影' 0 t
ajax83="ys_dy_list_actor_168580" >刘亦菲</a></em> <em><a href="http://dianying.2345.com/list/---liuyifei---.html"
target="_blank" data-ajax83="ys_dy_list_actor_168580" >吴亦凡</a></em></span>
    </li>
```

图 6-60　页面源代码

直接找<li>标签就可以了。先找<ul>标签，然后再嵌套查找<li>标签也行，更加精确。现在爬虫所有的要素都已经完备了，可以构造爬虫了。

6.6.2　项目实施

在 Eclipse 中创建新项目 movieBS4，然后在项目中创建新的 PyDev Module 文件 get2016movie.py，再将上面几节都用到过的 mylog.py 复制到 movieBS4 项目下。mylog.py 还是直接略过，主要是 get2016movie.py。

【示例 6-11】get2016movie.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月13日

@author: hstking hstking@hotmail.com
'''

from bs4 import BeautifulSoup
import urllib2
import codecs
from mylog import MyLog as mylog


class MovieItem(object):
    movieName = None
```

```
    movieScore = None
    movieStarring = None

class GetMovie(object):
    def __init__(self):
        self.urlBase = 'http://dianying.2345.com/list/----2016---1.html'
        self.log = mylog()
        self.pages = self.getPages()
        self.urls = []
        self.items = []
        self.getUrls(self.pages)
        self.spider(self.urls)
        self.pipelines(self.items)

    def getPages(self):
        self.log.info(u'开始获取页数')
        htmlContent = self.getResponseContent(self.urlBase)
        soup = BeautifulSoup(htmlContent, 'lxml')
        tag = soup.find('div', attrs={'class':'video_page'})
        subTags = tag.find_all('a')
        self.log.info(u'获取页数成功')
        return int(subTags[-2].get_text())

    def getResponseContent(self, url):
        try:
            response = urllib2.urlopen(url)
        except:
            self.log.error(u'Python 返回 URL:%s  数据失败' %url)
        else:
            self.log.info(u'Python 返回 URUL:%s  数据成功' %url)
            return response.read()

    def getUrls(self, pages):
        urlHead = 'http://dianying.2345.com/list/----2016---'
        urlEnd = '.html'
        for i in xrange(1,pages + 1):
            url = urlHead + str(i) + urlEnd
            self.urls.append(url)
            self.log.info(u'添加 URL:%s 到 URLS 列表' %url)

    def spider(self, urls):
        for url in urls:
            htmlContent = self.getResponseContent(url)
```

```
            soup = BeautifulSoup(htmlContent, 'lxml')
            anchorTag = soup.find('ul', attrs={'class':'globalPicTxt pic140 clearfix'})
            tags = anchorTag.find_all('li')
            for tag in tags:
                item = MovieItem()
                item.movieName = tag.find('span', attrs={'class':'sTit'}).get_text()
                item.movieScore = tag.find('span', attrs={'class':'pRightBottom'}).em.get_text().replace(u'分', '')
                item.movieStarring = tag.find('span', attrs={'class':'sDes'}).get_text().replace(u'主演：', '')
                self.items.append(item)
                self.log.info(u'获取电影名为：<<%s>>成功' %(item.movieName))

    def pipelines(self, items):
        fileName = u'2016热门电影.txt'.encode('GBK')
        with codecs.open(fileName, 'w', 'utf8') as fp:
            for item in items:
                fp.write('%s \t %s \t %s \r\n' %(item.movieName, item.movieScore, item.movieStarring))
                self.log.info(u'电影名为：<<%s>>已成功存入文件"%s"...' %(item.movieName, fileName.decode('GBK')))



if __name__ == '__main__':
    GM = GetMovie()
```

单击 Eclipse 图标栏的运行图标，顺利获取了所需的数据，如图 6-61 所示。

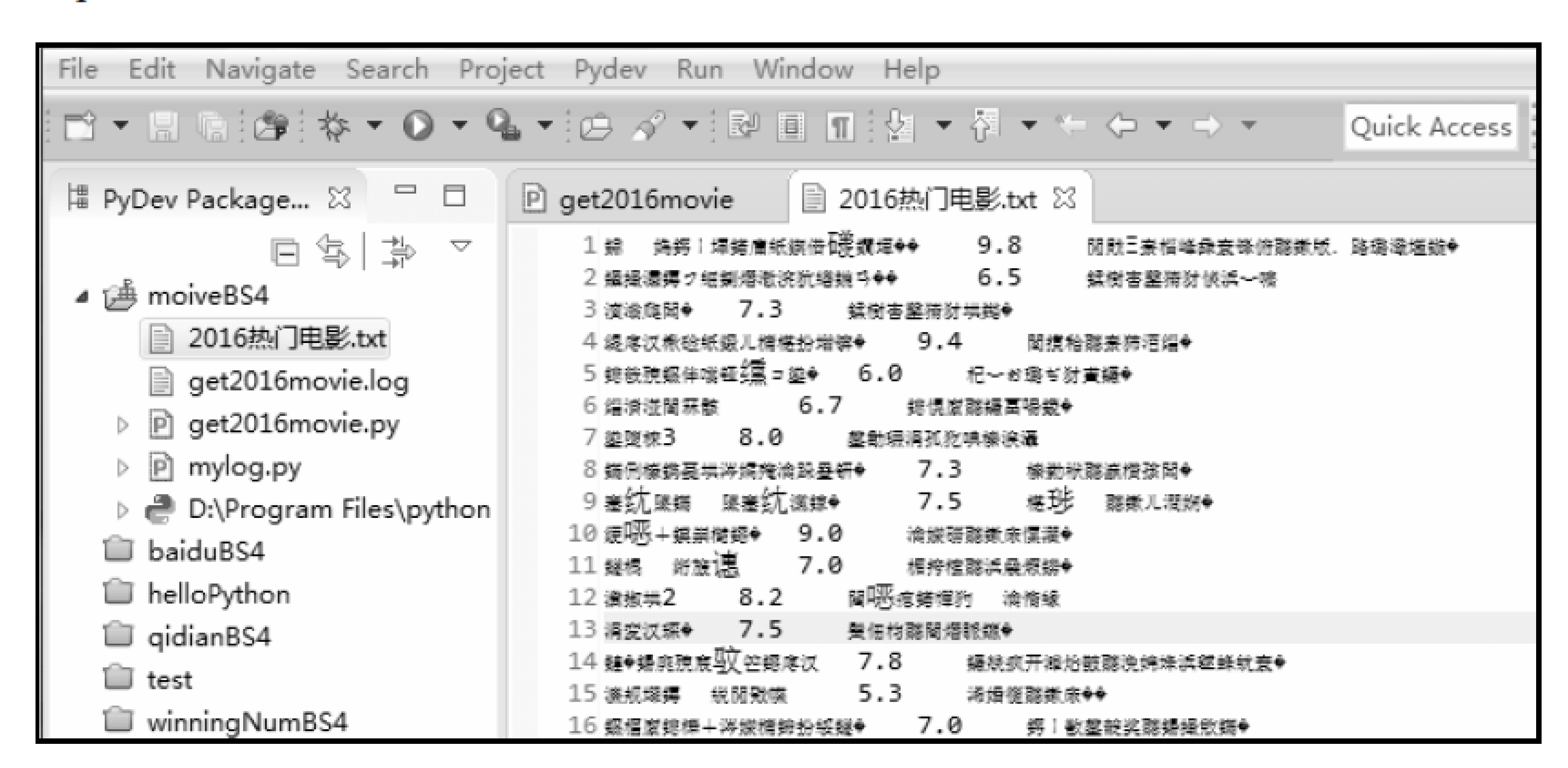


图 6-61 获取数据

这个爬虫做到这里就可以结束了。可这跟上节爬起点中文网站的爬虫太相似了，根本没

必要重复做一种爬虫吧。

6.6.3 bs4 反爬虫

前面曾经提到过，有的网站会因为爬虫频繁地爬取而被反爬虫程序封锁。万一哪天运气不好被封锁了怎么办？最简单的方法当然是换个 IP 再去爬，如果网站禁止机器人爬取，也简单，换个 headers 就可以了（一般爬虫默认的 user-agent 都比较特别，很容易被反爬虫程序找出来）。还记得每个项目中都有的 getResponseContent 函数吗？本来只需要一行就能解决的问题，每次都把它扩展成了一个函数，就是在这个时候用的。

【示例 6-12】将 getResponseConteng 函数修改一下，最终的 get2016movie.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月13日

@author: hstking hstking@hotmail.com
'''

from bs4 import BeautifulSoup
import urllib2
import codecs
from mylog import MyLog as mylog


class MovieItem(object):
    movieName = None
    movieScore = None
    movieStarring = None

class GetMovie(object):
    '''获取电影信息 '''
    def __init__(self):
        self.urlBase = 'http://dianying.2345.com/list/----2016---1.html'
        self.log = mylog()
        self.pages = self.getPages()
        self.urls = []  #url池
        self.items = []
        self.getUrls(self.pages) #获取抓取页面的url
        self.spider(self.urls)
        self.pipelines(self.items)

```

```
    def getPages(self):
        '''获取总页数 '''
        self.log.info(u'开始获取页数')
        htmlContent = self.getResponseContent(self.urlBase)
        soup = BeautifulSoup(htmlContent, 'lxml')
        tag = soup.find('div', attrs={'class':'video_page'})
        subTags = tag.find_all('a')
        self.log.info(u'获取页数成功')
        return int(subTags[-2].get_text())

    def getResponseContent(self, url):
        '''获取页面返回的数据 '''
        fakeHeaders= {'User-Agent':'Mozilla/5.0 (Windows NT 6.2; rv:16.0) Gecko/20100101 Firefox/16.0'}
        request = urllib2.Request(url.encode('utf8'), headers=fakeHeaders)

        proxy = urllib2.ProxyHandler({'http':'http://192.168.2.99:1080'})
        opener = urllib2.build_opener(proxy)
        urllib2.install_opener(opener)
        try:
            response = urllib2.urlopen(request)
        except:
            self.log.error(u'Python 返回 URL:%s  数据失败' %url)
        else:
            self.log.info(u'Python 返回 URUL:%s  数据成功' %url)
            return response.read()

    def getUrls(self, pages):
        urlHead = 'http://dianying.2345.com/list/----2016---'
        urlEnd = '.html'
        for i in xrange(1,pages + 1):
            url = urlHead + str(i) + urlEnd
            self.urls.append(url)
            self.log.info(u'添加 URL:%s 到 URLS 列表' %url)

    def spider(self, urls):
        for url in urls:
            htmlContent = self.getResponseContent(url)
            soup = BeautifulSoup(htmlContent, 'lxml')
            anchorTag = soup.find('ul', attrs={'class':'globalPicTxt pic140 clearfix'})
            tags = anchorTag.find_all('li')
            for tag in tags:
```

```
                item = MovieItem()
                item.movieName = tag.find('span', attrs={'class':'sTit'}).get_text()
                item.movieScore = tag.find('span', attrs={'class':'pRightBottom'}).em.get_text().replace(u'分', '')
                item.movieStarring = tag.find('span', attrs={'class':'sDes'}).get_text().replace(u'主演：', '')
                self.items.append(item)
                self.log.info(u'获取电影名为：<<%s>>成功' %(item.movieName))

    def pipelines(self, items):
        fileName = u'2016 热门电影.txt'.encode('GBK')
        with codecs.open(fileName, 'w', 'utf8') as fp:
            for item in items:
                fp.write('%s \t %s \t %s \r\n' %(item.movieName, item.movieScore, item.movieStarring))
                self.log.info(u'电影名为：<<%s>>已成功存入文件"%s"...' %(item.movieName, fileName.decode('GBK')))



if __name__ == '__main__':
    GM = GetMovie()
```

好了，这样再也不怕被网站封锁了。封锁一次，大不了再换个代理而已，简简单单解决问题。

6.6.4　代码分析

本节的代码与上节的起点网站爬虫很相似，这里就不详细解析了，只分析一下其中的不同之处。第一个不同之处在于模块的导入，本节的 get2016movie.py 导入了 codecs 模块。这个模块可以选择输入字符的编码。之前的程序在写入 txt 时都需要将字符串的编码转换成 utf8。这里只需要用 codecs.open(filename, ‘w’, ‘utf8’)打开文件就可以了。后面往句柄中输入的字符串都会自动保存为 utf8 的编码。

第二个不同就是 getResponseContent 函数了。本节在 getResponseContent 函数中添加了一个浏览器的 headers 和一个经过验证、可以使用的 proxy，解除了网站反爬虫程序的封锁。但这种反爬虫很被动，只有被封锁了才会解除封锁，而不能主动避免反爬虫程序的封锁。

6.7 bs4 爬虫实战五：获取音悦台榜单

既然要反反爬虫，那就要反个彻底。在 Scrapy 里曾提过反爬虫的运行机制，基本上就是通过 IP、headers 来锁定爬虫用户，然后进行封锁（用验证码、验证图案的不在讨论范围中）。前面的 Scrapy 使用的是随机跳转 proxy 和 headers 的方法对付反爬虫。这里也是如此（反反爬虫的手段远不止这些，比如使用专门网站爬虫、分布式爬虫都可以。个人用户没什么特殊要求，做到这一步就差不多了）。

6.7.1 目标分析

本节将使用随机 proxy 和 headers 主动抵抗反爬虫。打开 www.yinyuetai.com 网站，本节爬虫的目的是爬取音悦台网站公布的 MV 榜单。单击网站最上方的“V 榜”，从弹出菜单中选取“MV 作品榜”选项，打开了音乐 V 榜。以内地篇为例，网站排列除了内地 MV 音乐榜的前 50 名，使用了 3 个网页。这 3 个页面的网址分别为 http://vchart.yinyuetai.com/vchart/trends?area=ML&page=1、http://vchart.yinyuetai.com/vchart/trends?area=ML&page=2、http://vchart.yinyuetai.com/vchart/trends?area=ML&page=3，如图 6-62 所示。

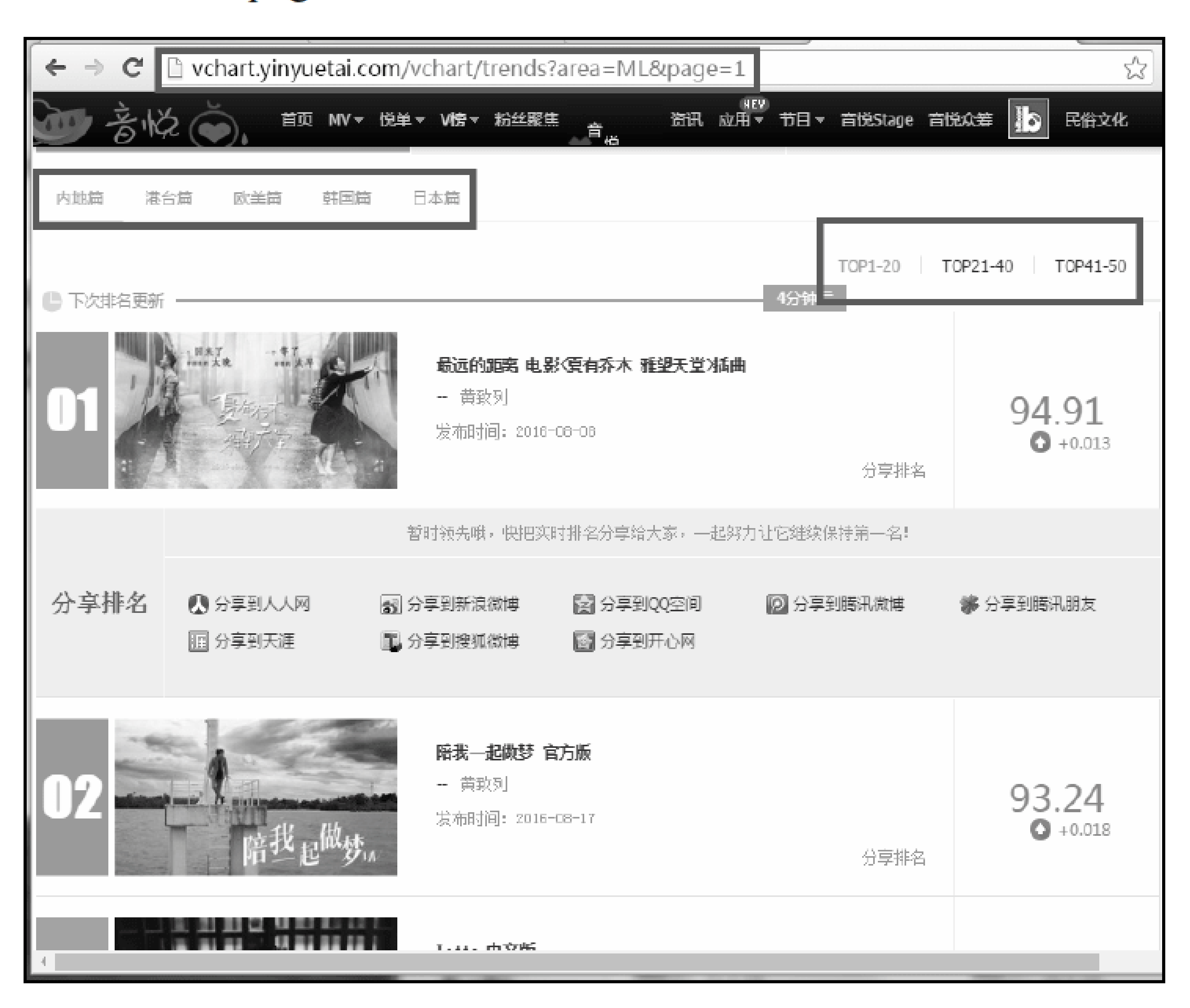


图 6-62 音悦台榜单

看看其他几个 V 榜中的地区代码，分别是 HT、US、KR 和 JP，Urls 的规则很明了了。再来看看爬虫的抓取规则。在网页的任意空白处右击，选择弹出菜单中的“查看网页源代码”项，打开页面源代码页面，如图 6-63 所示。

```
view-source:vchart.yinyuetai.com/vchart/trends?area=HT
                                                                  top_num
            <p class="asc-data clearfix">
                <em class="score-asc"></em>
                <span class="asc-num">0.000</span>
            </p>


        </div>
        <div class="top_num">01</div>
        <div class="mv_info">
            <div class="mv_info_head_img_container J_add_convenient_container">
                <a class="img video-bo-img" href="http://v.yinyuetai.com/video/2646605" target="_b
                    <img src="http://img4.yytcdn.com/video/mv/160811/2646605/-M-5cae6718c88891b2a
alt="终于 官方版"/>

                    <div class="bo-mask">
                        <i class="bo-icon"></i>
                    </div>
                </a>
                        <span class="J_add_convenient" title="加入便捷悦单" data-video-id="2646605
                            style="display: none;"></span>
            </div>

            <div class="info">
                <h3>
                    <a target="_blank" href="http://v.yinyuetai.com/video/2646605">终于 官方版</a>

                </h3>

                <p class="cc"><em>--</em>


                        <a class="special" href="http://www.yinyuetai.com/fanclub/168"
                          target="_blank">张韶涵</a>

                </p>

                <p class="c9">发布时间: 2016-08-12</p>

                <div class="dataes">

                </div>
```

图 6-63　源码页面

所有的上榜 MV 都在标签<div class="op_num">这个标签下。爬虫的抓取规则也有了，下面就看具体实施了。

6.7.2　项目实施

打开 Eclipse，创建新项目 YinYueTaiBS4URL，并在项目中创建一个 PyDev Modules 文件 getTrendsMV.py 作为主文件，把上节项目中使用过的 mylog.py 复制到当前目录下。因为要使用不同的 proxy 和 headers，再创建一个新的资源文件 resource.py。

老规矩，mylog.py 可以略过，这次先看看 resource.py。

【示例 6-13】resource.py 的内容如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'

UserAgents = [
 "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; AcooBrowser; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",
 "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506)",
 "Mozilla/4.0 (compatible; MSIE 7.0; AOL 9.5; AOLBuild 4337.35; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)",
 "Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)",
```

```
"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 2.0.50727; Media Center PC 6.0)",
"Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 1.0.3705; .NET CLR 1.1.4322)",
"Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 5.2; .NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.2; .NET CLR 3.0.04506.30)",
"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN) AppleWebKit/523.15 (KHTML, like Gecko, Safari/419.3) Arora/0.3 (Change: 287 c9dfb30)",
"Mozilla/5.0 (X11; U; Linux; en-US) AppleWebKit/527+ (KHTML, like Gecko, Safari/419.3) Arora/0.6",
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2pre) Gecko/20070215 K-Ninja/2.1.1",
"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9) Gecko/20080705 Firefox/3.0 Kapiko/3.0",
"Mozilla/5.0 (X11; Linux i686; U;) Gecko/20070322 Kazehakase/0.4.5",
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.8) Gecko Fedora/1.9.0.8-1.fc10 Kazehakase/0.5.6",
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11",
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/535.20 (KHTML, like Gecko) Chrome/19.0.1036.7 Safari/535.20",
"Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; fr) Presto/2.9.168 Version/11.52",
]

PROXIES = [
'58.20.238.103:9797',
'123.7.115.141:9797',
'121.12.149.18:2226',
'176.31.96.198:3128',
'61.129.129.72:8080',
'115.238.228.9:8080',
'124.232.148.3:3128',
'124.88.67.19:80',
'60.251.63.159:8080',
'118.180.15.152:8102'
]
```

这个文件很简单，仅包含了 2 个列表。UserAgents 列表只需要在网上找一下，可以找到各种各样的 headers，排除掉移动端的也有不少（有的网站根据客户端的不同返回的数据也不同），尽量选择大众化的 UserAgent。而 proxies 那就更简单了，Scrapy 项目中不是有个

getProxy 项目吗？执行条命令而已，来个 100 条 proxy 都没问题。这里仅为测试给数十个 proxy 就可以了，只需稍微注意一下，不要选择 https 协议的就可以了（https 协议的 proxy 也可以用，但会增加代码工作量。既然 http 协议的够用了，就不用那么费劲了）。

【示例 6-14】主程序 getTrendsMV.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月10日

@author: hstking hstking@hotmail.com
'''

from bs4 import BeautifulSoup
import urllib2
import codecs
import time
from mylog import MyLog as mylog
import resource
import random

class Item(object):
    top_num = None  #排名
    score = None  #打分
    mvName = None  #MV 名字
    singer = None  #演唱者
    releasTime = None  #释放时间


class GetMvList(object):
    '''The all data from www.yinyuetai.com
    所有数据都来自 www.yinyuetai.com
    '''
    def __init__(self):
        self.urlBase = 'http://vchart.yinyuetai.com/vchart/trends?'
        self.areasDic = {'ML':'Mainland','HT':'Hongkong&Taiwan','US':'Americ','KR':'Korea','JP':'Japan'}
        self.log = mylog()
        self.getUrls()


    def getUrls(self):
```

```
        '''获取 url 池 '''
        areas = ['ML','HT','US','KR','JP']
        pages = [str(i) for i in range(1,4)]
        for area in areas:
            urls = []
            for page in pages:
                urlEnd = 'area=' + area + '&page=' + page
                url = self.urlBase + urlEnd
                urls.append(url)
                self.log.info(u'添加 URL:%s 到 URLS' %url)
            self.spider(area, urls)



    def getResponseContent(self, url):
        '''从页面返回数据 '''
        fakeHeaders = {'User-Agent':self.getRandomHeaders()}
        request = urllib2.Request(url.encode('utf8'), headers=fakeHeaders)

        proxy =
urllib2.ProxyHandler({'http':'http://'+self.getRandomProxy()})
        opener = urllib2.build_opener(proxy)
        urllib2.install_opener(opener)
        try:
            response = urllib2.urlopen(request)
            time.sleep(1)
        except:
            self.log.error(u'Python 返回 URL:%s  数据失败' %url)
            return ''
        else:
            self.log.info(u'Python 返回 URUL:%s  数据成功' %url)
            return response.read()

    def spider(self,area,urls):
        items = []
        for url in urls:
            responseContent = self.getResponseContent(url)
            if not responseContent:
                continue
            soup = BeautifulSoup(responseContent, 'lxml')
            tags = soup.find_all('li', attrs={'name':'dmvLi'})
            for tag in tags:
                item = Item()
```

```
                item.top_num = tag.find('div', attrs={'class':'top_num'}).get_text()
                if tag.find('h3', attrs={'class':'desc_score'}):
                    item.score = tag.find('h3', attrs={'class':'desc_score'}).get_text()
                else:
                    item.score = tag.find('h3', attrs={'class':'asc_score'}).get_text()

                item.mvName = tag.find('img').get('alt')
                item.singer = tag.find('a', attrs={'class':'special'}).get_text()
                item.releaseTime = tag.find('p', attrs={'class':'c9'}).get_text()
                items.append(item)
                self.log.info(u'添加 mvName 为<<%s>>的数据成功' %(item.mvName))
        self.pipelines(items, area)


    def getRandomProxy(self):  #随机选取 proxy 代理
        return random.choice(resource.PROXIES)

    def getRandomHeaders(self):  #随机选取文件头
        return random.choice(resource.UserAgents)


    def pipelines(self, items, area):  #处理获取的数据
        fileName = 'mvTopList.txt'
        nowTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())
        with codecs.open(fileName, 'a', 'utf8') as fp:
            fp.write('%s -------%s\r\n' %(self.areasDic.get(area), nowTime))
            for item in items:
                fp.write('%s %s \t %s \t %s \t %s \r\n'
                         %(item.top_num, item.score, item.releaseTime, item.singer, item.mvName))
                self.log.info(u'添加 mvName 为<<%s>>的 MV 到%s...' %(item.mvName, fileName))
            fp.write('\r\n'*4)


if __name__ == '__main__':
    GML = GetMvList()
```

所有的代码都完成了，开始运行。单击 Eclipse 图标栏上的运行图标，执行 getTrendsMV.py，得到的结果如图 6-64 所示。

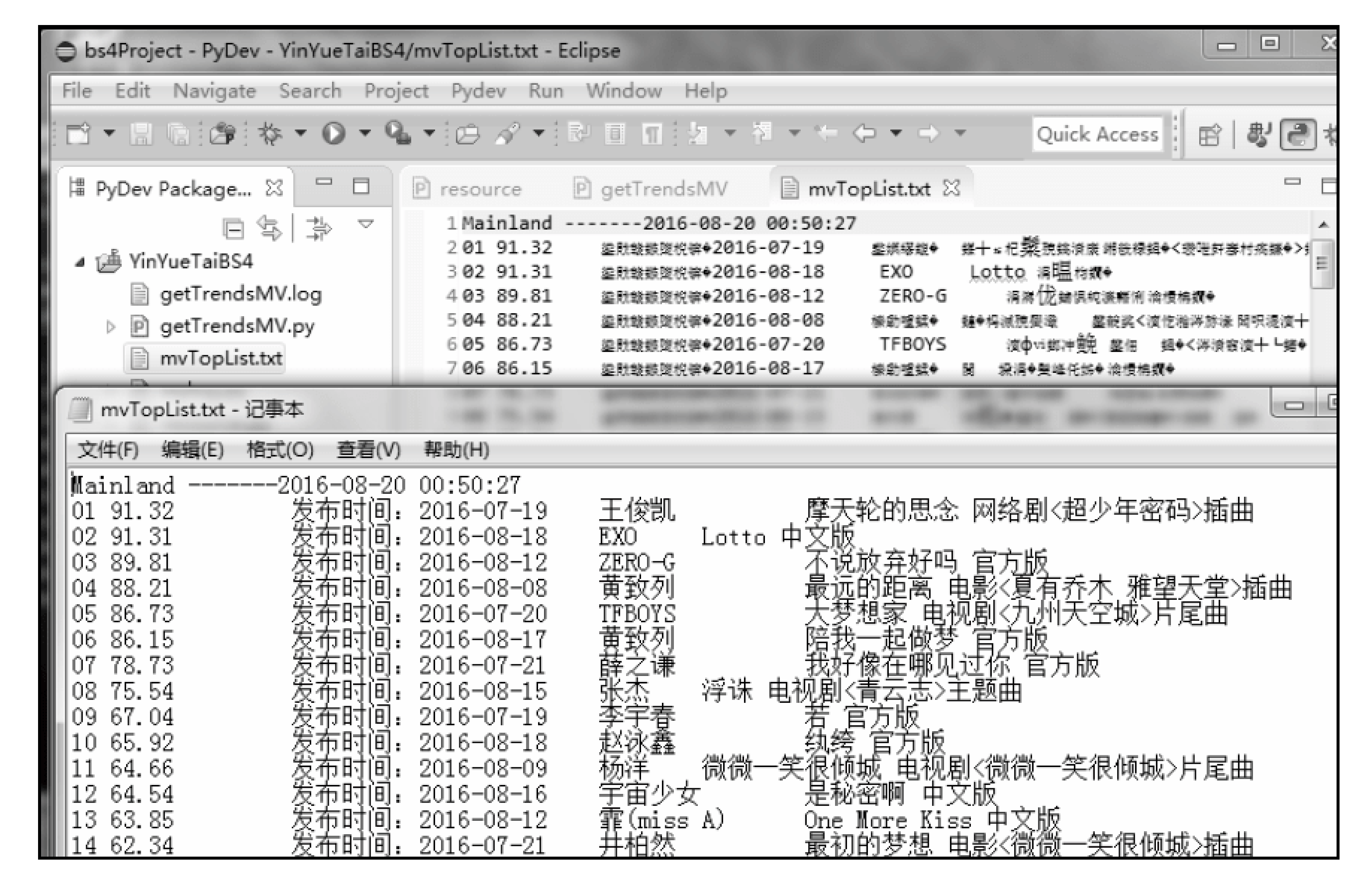


图 6-64 run getTrendsMV.py

好了，程序执行结果没问题。如果想知道实时榜单，单击鼠标运行一下程序就可以了。

6.7.3 代码分析

本节的 getTrendsMV.py 爬虫从功能上看要比上节的被动式反爬虫要复杂一些，但代码更简单一点。项目中 mylog.py 没什么好说的，就一 log 助手而已。resource.py 也无须多说，就一资源文件。如果觉得手动挑选 proxy 比较麻烦，可以再写一个 Python 程序让其自动导入 proxy 到 resource.py 中。唯一需要分析的就是 getTrendsMV.py 主程序了。

第 9~15 行，开头还是导入所需的模块，这里只是多导入了一个 random 模块，用于从列表中随机挑选 User-Agent 和 proxy。

第 17~22 行创建一个 Item 类，这个是仿照 Scrapy 的 Item.py 写的，也很简单。

第 25~106 行是 GetMvList 类。这个类将从 yinyuetai 网站中获取所需的数据。第 29~32 行给出了起始页面和 urls 的规则列表，然后__init__函数自动执行了 self.getUrls 函数。self.getUrls 再调用其他函数，完成类设计的功能。第 36~47 行是 getUrls 函数，将利用起始页 url 的页面规则，将所有需要爬取的网页 url 存入到 urls 列表中。第 51~67 行是 getResponseContent 函数，它将返回请求 url 的数据。每执行一次 getResponseContent 函数都将调用一次 self.getRandomHeaders 和 self.getRandomProxy 函数。随机选择一个 proxy 和 User-Agent 使用，使网站的反爬虫工具无从捕捉。第 61 行的作用是每执行一次函数都将暂停 1 秒，避免被反爬虫工具发觉。第 69~90 行是 spider 函数，它的作用就是根据爬虫的抓取规则，从返回的数据中抓取所需的数据。第 93~97 行是 getRandomProxy 函数和

getRandomHeaders 函数，虽然只有 1 行命令，但还是写成了函数，这是为了以后有什么新功能可以很方便地添加进去。第 100~109 行是 pipelines 函数，将所有抓取到的有效数据保存到 txt 文件中。第 112~113 行是程序的__main__程序，只有一条命令，用于实例化 GetMvList 类。

6.8 本章小结

Bs4 爬虫也很强大。它的优点在于可以随心所欲地定制爬虫，缺点就是稍微复杂了一点点，需要从头到尾的写代码，这毕竟是作文题。填空题虽然简单，但从头到尾都身不由己，得按照框架作者的思路走。作文题毕竟是自己写的，可以随心所欲地修改调整。如果是比较小的项目，个人建议还是用 bs4，可以有针对性地根据自己的需要编写爬虫。大项目，那还是建议选 Scrapy 吧，Scrapy 能流行至今可不是浪得虚名。

第 7 章

Mechanize模拟浏览器

细心的读者应该已经发现了，本章之前的爬虫讲的都是对一般静态网站的数据过滤，但不是所有的网站都可以这样简单地得到数据的。比如，某些站点需要登录后才能获取数据，这样一来，仅靠 urllib2 模块就有点力不从心了（urllib2 模块也可以爬取动态网站的数据，不过过程就很麻烦了）。幸好 Python 还有更加强力的模块 Mechanize。

Mechanize 并不是爬虫，它是一个 Python 模块，用于模拟浏览器的模块。本书讲的是网络爬虫，直接爬数据就好了，为什么会跟 Mechanize 扯上关系呢？前面的章节都是使用 urllib2 模块向服务器发送请求的。如果网页要求登录，输入用户名、密码，urllib2 可以应付。但如果是需要输入验证码那就麻烦了。目前是有开源的方案可以解决这个问题，只不过需要绕很多的弯路。倒不如直接使用模拟浏览器，很方便地解决这个问题。

Python 的第三方模块中，能模拟浏览器的模块也不少。选择 Mechanize 的原因在于它的易用性和实用性比较平衡，功能强大而又简单易用，就选它了。

7.1 安装 Mechanize 模块

Mechanize 的官网是 http://wwwsearch.sourceforge.net/mechanize/。在官网中给出了 3 种安装方法：Easy_install 安装、源码安装和 git 安装。实际安装时可以根据平台特性选择最简单的安装方法即可。

7.1.1 Windows 下安装 Mechanize

在 Windows 中安装 Python 的第三方模块，最简单的方法莫过于 pip 了（前提条件是已经配置过 pip 源了，使用初始源会比较慢）。打开 cmd.exe，执行命令：

```
pip install mechanize
```

执行结果如图 7-1 所示。

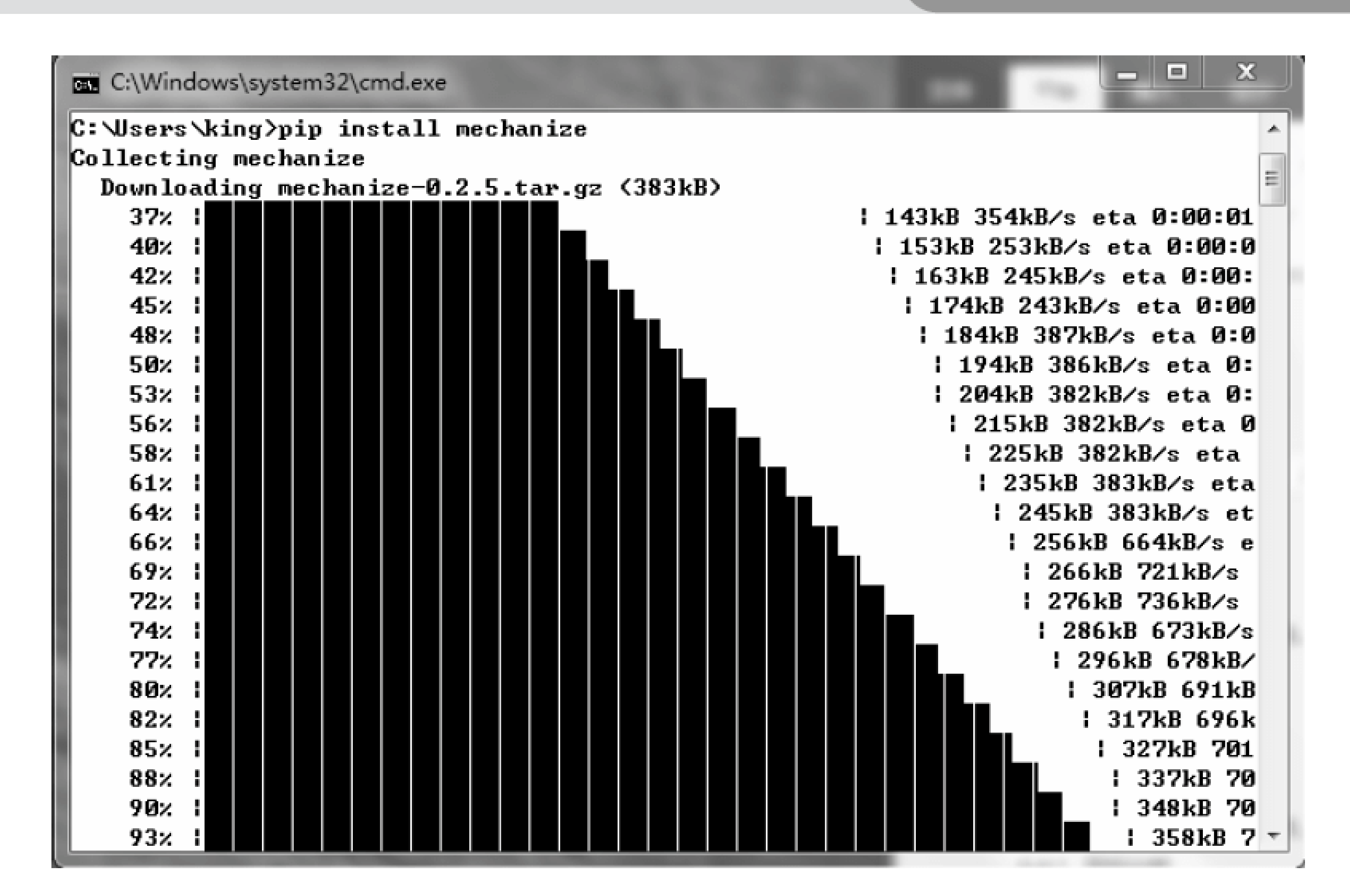


图 7-1　Windows 安装 mechanize

已经把 Mechanize 安装到 Windows 上，可以直接使用了。

7.1.2　Linux 下安装 Mechanize

在 Linux 找安装软件最简单的方法还是 apt-get，感谢“万能”的 deiban 软件库，即使是 Python 模块也可以用 apt-get 一键安装。不必介意 Mechanize 官网上的安装建议，怎么简单怎么来就可以了。执行命令：

```
apt-get install python-mechanize
```

执行结果如图 7-2 所示。

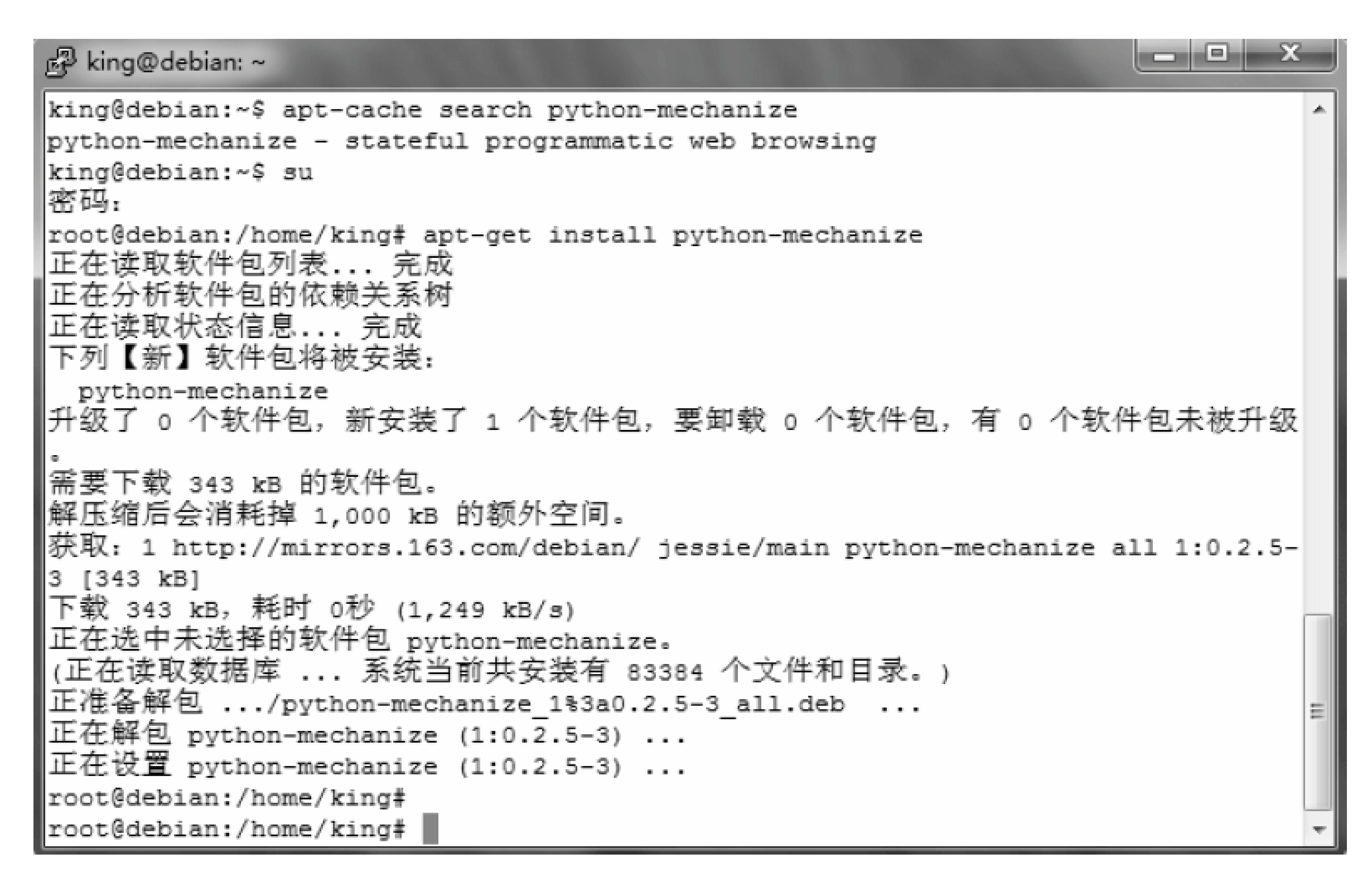


图 7-2　Linux 安装 mechanize

因为 Mechanize 很久没更新的缘故，Linux 下安装的最稳定版本也是最新的版本 0.2.5。

不过无须担忧，Mechanize 已经很强大了。使劲地往好的方面想，说不定没更新是因为这个模块已经改无可改了呢？

7.2 Mechanize 测试

百闻不如一见，说得再多也不如直接测试一次。Mechanize 模块，常用的命令、方法并不多，作为一般使用者，无须追求掌控所有细节，只需要能使用、会使用即可。它只是个很简单的模块，多试几次就能熟练掌握。

7.2.1 Mechanize 百度

先试下最简单的用法，以最常用的网站百度为例。使用 Mechanize 访问百度搜索站点，并使用百度搜索“python 网路爬虫”得到返回结果。如果不使用 Mechanize，那就只能在浏览器中输入搜索的关键字，再观察 URL 的变化规律，最后将所有的 URL 注入到列表中，一个个地返回结果爬取数据。下面演示如何使用 Mechanize 模拟浏览器，搜索关键字。

使用 putty 连接到 Linux，运行 Python 程序，并导入 Mechanize 模块，如图 7-3 所示。

```
king@debian: ~
king@debian:~$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import mechanize
>>>
```

图 7-3 mechanize 环境

在 Python 环境下，执行命令：

```
    br = mechanize.Browser()
    br.set_handle_equiv(True)
    br.set_handle_redirect(True)
    br.set_handle_referer(True)
    br.set_handle_robots(False)
    br.set_handle_gzip(False)
    br.set_handle_refresh(mechanize._http.HTTPRefreshProcessor(), max_time=1)
    br.addheaders = [('User-agent', 'Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.9.0.1) Gecko/2008071615 Fedora/3.0.1-1.fc9 Firefox/3.0.1')]
```

执行结果如图 7-4 所示。

```
king@debian: ~
king@debian:~$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import mechanize
>>> br = mechanize.Browser()
>>> br.set_handle_equiv(True)
>>> br.set_handle_redirect(True)
>>> br.set_handle_referer(True)
>>> br.set_handle_robots(False)
>>> br.set_handle_gzip(False)
>>> br.set_handle_refresh(mechanize._http.HTTPRefreshProcessor(), max_time=1)
>>> br.addheaders = [('User-agent', 'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:
1.9.0.1) Gecko/2008071615 Fedora/3.0.1-1.fc9 Firefox/3.0.1')]
>>>
```

图 7-4　Browser 环境设置

使用 mechanize 浏览器打开百度搜索的主页，并查看打开网页中的框架。执行命令：

```
br.open('https://www.baidu.com'):
for form in br.forms():
 print form
```

执行结果如图 7-5 所示。

```
king@debian: ~
>>>
>>> br.open('https://www.baidu.com')
<response_seek_wrapper at 0x7fa5b4c91e60 whose wrapped object = <closeable_respo
nse at 0x7fa5b4c91368 whose fp = <socket._fileobject object at 0x7fa5b4cccc50>>>
>>> for form in br.forms():
...     print form
...
<f GET https://www.baidu.com/s application/x-www-form-urlencoded
  <HiddenControl(ie=utf-8) (readonly)>
  <HiddenControl(f=8) (readonly)>
  <HiddenControl(rsv_bp=1) (readonly)>
  <HiddenControl(rsv_idx=1) (readonly)>
  <HiddenControl(ch=) (readonly)>
  <HiddenControl(tn=baidu) (readonly)>
  <HiddenControl(bar=) (readonly)>
  <TextControl(wd=)>
  <SubmitControl(<None>=百度一下) (readonly)>
  <HiddenControl(rn=) (readonly)>
  <HiddenControl(oq=) (readonly)>
  <HiddenControl(rsv_pq=e21416f7000ac4be) (readonly)>
  <HiddenControl(rsv_t=2358R/NVmDD+/zEba6aqmOPkH9Q2oHj304sMNarrDF8C+yxoLNtKDM4Oz
qk) (readonly)>
  <HiddenControl(rqlang=cn) (readonly)>>
>>>
```

图 7-5　显示网页框架

从图 7-5 中可以看出，只有一个名字为 f 的框架（有时候框架没有名字，那就只能用它们的顺序来选择。第一个框架是 nr=0，第二个是 nr=1，以此类推）。输入文字的位置为文本输入框<TextControl (wd=)>。选择框架，在框架内输入数据后提交数据。以搜索“Python 网络爬虫”为例，执行命令：

```
br.select_form(name='f')
br.form['wd'] = 'Python 网络爬虫'
br.submit()
```

执行结果如图 7-6 所示。

```
>>>
>>> br.select_form(name='f')
>>> br.form['wd'] = 'Python 网络爬虫'
>>> br.submit()
<response_seek_wrapper at 0x7fa5b4c91dd0 whose wrapped object = <closeable_respo
nse at 0x7fa5b4cba830 whose fp = <socket._fileobject object at 0x7fa5b4cd3dd0>>>
>>> 
```

图 7-6　搜索关键字

返回搜索的结果，执行命令：

```
print br.response().read()
```

执行结果如图 7-7 所示。



```
king@debian: ~

<div class="c-row c-gap-top">

    <div class="c-span4  opr-recommends-merge-item opr-recommends-merge-item-ver
tical" data-click="{'rsv_re_ename':'python计算与编程实践','rsv_re_uri':'c50e36f1
f0084ec6bd908f5ea30d1be7'}">
                                                            <div class="opr-recommen
ds-merge-p">
            <a target="_blank" href="/s?rsv_idx=1&wd=python%E8%AE%A1%E7%AE%97%E4
%B8%8E%E7%BC%96%E7%A8%8B%E5%AE%9E%E8%B7%B5&usm=1&ie=utf-8&rsv_cq=python+%E7%BD%9
1%E7%BB%9C%E7%88%AC%E8%99%AB&rsv_dl=0_right_recommends_merge_21102&amp;cq=python
%20%E7%BD%91%E7%BB%9C%E7%88%AC%E8%99%AB&amp;srcid=28310&amp;rt=%E8%AE%A1%E7%AE%9
7%E6%9C%BA%E7%B1%BB%E4%B9%A6%E7%B1%8D&amp;recid=21102&amp;euri=c50e36f1f0084ec6b
d908f5ea30d1be7"><img  data-img="https://ss2.baidu.com/6ONYsjip0QIZ8tyhnq/it/u=4
063802933,465813&fm=58" class="c-img c-img4 opr-recommends-merge-img"/></a>
                                </div>
        <div class="c-gap-top-small"><a target="_blank" title="python计算与编程
实践" href="/s?rsv_idx=1&wd=python%E8%AE%A1%E7%AE%97%E4%B8%8E%E7%BC%96%E7%A8%8B%
E5%AE%9E%E8%B7%B5&usm=1&ie=utf-8&rsv_cq=python+%E7%BD%91%E7%BB%9C%E7%88%AC%E8%99
%AB&rsv_dl=0_right_recommends_merge_21102&amp;cq=python%20%E7%BD%91%E7%BB%9C%E7%
88%AC%E8%99%AB&amp;srcid=28310&amp;rt=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%B1%BB%E4%B9
%A6%E7%B1%8D&amp;recid=21102&amp;euri=c50e36f1f0084ec6bd908f5ea30d1be7">python计
算与编程实践</a></div>
                                    </div>
```

图 7-7　返回搜索结果

查看返回页面的所有链接，执行命令：

```
for link in br.links():
 print("%s : %s" %(link.url, link.text))
```

执行结果如图 7-8 所示。

```
king@debian: ~
http://tieba.baidu.com/f?kw=Python%20%CD%F8%C2%E7%C5%C0%B3%E6&fr=wwwt : 贴吧
http://zhidao.baidu.com/q?ct=17&pn=0&tn=ikaslist&rn=10&word=Python%20%CD%F8%C2%E
7%C5%C0%B3%E6&fr=wwwt : 知道
http://music.baidu.com/search?fr=ps&ie=utf-8&key=Python%20%CD%F8%C2%E7%C5%C0%B3%
E6 : 音乐
http://image.baidu.com/search/index?tn=baiduimage&ps=1&ct=201326592&lm=-1&cl=2&n
c=1&ie=utf-8&word=Python%20%CD%F8%C2%E7%C5%C0%B3%E6 : 图片
http://v.baidu.com/v?ct=301989888&rn=20&pn=0&db=0&s=25&ie=utf-8&word=Python%20%C
D%F8%C2%E7%C5%C0%B3%E6 : 视频
http://map.baidu.com/m?word=Python%20%CD%F8%C2%E7%C5%C0%B3%E6&fr=ps01000 : 地图
http://wenku.baidu.com/search?word=Python%20%CD%F8%C2%E7%C5%C0%B3%E6&lm=0&od=0&i
e=utf-8 : 文库
//www.baidu.com/more/ : 更多»
javascript:; : 展开
/s?rsv_idx=1&wd=%E8%87%AA%E5%B7%B1%E5%8A%A8%E6%89%8B%E5%86%99%E7%BD%91%E7%BB%9C%
E7%88%AC%E8%99%AB&usm=1&ie=utf-8&rsv_cq=python+%E7%BD%91%E7%BB%9C%E7%88%AC%E8%99
%AB&rsv_dl=0_right_recommends_merge_21102&cq=python%20%E7%BD%91%E7%BB%9C%E7%88%A
C%E8%99%AB&srcid=28310&rt=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%B1%BB%E4%B9%A6%E7%B1%8D
&recid=21102&euri=b44b507f21c84b2991b738b574444d1b : [IMG]
/s?rsv_idx=1&wd=%E8%87%AA%E5%B7%B1%E5%8A%A8%E6%89%8B%E5%86%99%E7%BD%91%E7%BB%9C%
E7%88%AC%E8%99%AB&usm=1&ie=utf-8&rsv_cq=python+%E7%BD%91%E7%BB%9C%E7%88%AC%E8%99
%AB&rsv_dl=0_right_recommends_merge_21102&cq=python%20%E7%BD%91%E7%BB%9C%E7%88%A
C%E8%99%AB&srcid=28310&rt=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%B1%BB%E4%B9%A6%E7%B1%8D
&recid=21102&euri=b44b507f21c84b2991b738b574444d1b : 自己动手写网络爬虫
```

图 7-8　返回链接

使用 mechanize 浏览器打开指定链接，执行命令：

```
newLink = br.click_link(text='自己动手写网络爬虫')
br.open(newLink)
```

执行结果如图 7-9 所示。

```
>>>
>>> newLink = br.click_link(text='自己动手写网络爬虫')
>>> br.open(newLink)
<response_seek_wrapper at 0x7fa5b4d08200 whose wrapped object = <closeable_respo
nse at 0x7fa5b465edd0 whose fp = <socket._fileobject object at 0x7fa5b45e5050>>>
>>> 
```

图 7-9　打开链接

如果觉得打开的链接不对，还可以使用 br.back()命令返回上一个页面。Mechanize 的基本操作就是这些了。

7.2.2　Mechanize 光猫 F460

Mechanize 可以模拟登录，只是现在几乎所有的站点登录都需要输入验证码。虽然也有开源的解决方案，可以解决验证码什么的，但后面有更简单的解决方案，没必要在这里与验证码死磕。笔者一向认为，最简单的方法就是最合适的方法，宁愿多敲几行代码，也要选择最简单的。

如今无须验证码就可以登录的站点可不好找。好在身边有一个现成的 Web 服务器符合要求，F460 光猫的配置页面。而且还正好是动态回复数据的，简直是为 Mechanize 量身定做的。

在浏览器中打开光猫 F460 的配置页面 http://192.168.1.1，执行结果如图 7-10 所示。



图 7-10　F460 光猫登录

填写用户名和密码后单击“登录”框（用户名是 admin，密码要么直接问电信，要么在百度里搜索一下“hack f460 光猫”），进入配置界面，结果如图 7-11 所示。



图 7-11　获取光猫 F460 信息

先用 Python 模拟测试一次。打开 putty，登录到 Linux，进入到 Python 环境。执行命令：

```
import mechanize
cj = mechanize.CookieJar()
 br = mechanize.Browser()
 br.set_handle_equiv(True)
 br.set_handle_gzip(False)
 br.set_handle_redirect(True)
 br.set_handle_referer(True)
 br.set_handle_robots(False)
 br.set_handle_refresh(mechanize._http.HTTPRefreshProcessor(), max_time=1)
 br.addheaders = [('User-agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36')]
 br.set_cookiejar(cj)
 br.open('http://192.168.1.1')
```

执行结果如图 7-12 所示。

```
king@debian: ~

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Aug 25 15:41:52 2016 from 192.168.2.99
king@debian:~$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import mechanize
>>> cj = mechanize.CookieJar()
>>> br = mechanize.Browser()
>>> br.set_handle_equiv(True)
>>> br.set_handle_gzip(False)
>>> br.set_handle_redirect(True)
>>> br.set_handle_referer(True)
>>> br.set_handle_robots(False)
>>> br.set_handle_refresh(mechanize._http.HTTPRefreshProcessor(), max_time=1)
>>> br.addheaders = [('User-agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWe
bKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36')]
>>> br.set_cookiejar(cj)
>>> br.open('http://192.168.1.1')
<response_seek_wrapper at 0x7f10beffc908 whose wrapped object = <closeable_respo
nse at 0x7f10bf06dbd8 whose fp = <socket._fileobject object at 0x7f10bf032c50>>>
>>>
```

图 7-12　模拟浏览器打开光猫 f460 主页

查看网页上的框架，执行命令：

```
for form in br.forms():
 Print form
```

执行结果如图 7-13 所示。

```
>>>
>>> for form in br.forms():
...     print form
...
<fLogin POST http://192.168.1.1 application/x-www-form-urlencoded
  <TextControl(Username=)>
  <PasswordControl(Password=)>
  <SubmitControl(<None>=▒▒ ▒) (readonly)>
  <SubmitControl(<None>=▒▒ ▒▒) (readonly)>
  <HiddenControl(Frm_Logintoken=) (readonly)>>
>>>
```

图 7-13　查看主页框架

从图 7-13 可以得知，主页框架的名字是 fLogin，框架内文本框变量名是 Username，密码框的变量名是 Password。进入文本框，给变量赋值后，发送数据。执行命令：

```
br.select_form(name='fLogin')
br.form['Username'] = 'admin'
br.form['Password'] = '********'#这里输入光猫 F460 的密码
br.submit()
print br.response().read().decode('gb2312')
```

其中在选择框架时，可以用框架名字，也可以用框架的序列号，序列号从 0 开始。例如在这里选择框架时就可以用 br.select_form(nr=0)。如果需要选择第二个框架，则是 br.select_form(nr=1)。执行结果如图 7-14 所示。



图 7-14　获取框架 URL

这里显示了框架的链接。根据链接的地址 template.gch，直接使用 Mechanize 创建的浏览器打开这个链接就可以了。执行命令：

```
br.open('http://192.168.1.1/template.gch')
print br.response().read().decode('gb2312')
```

执行结果如图 7-15 所示。

```
king@debian: ~
<div class="space_0">
<table  class="infor" id="TABLE_DEV" width="410" border="0" cellpadding="0" cell
spacing="1" bgcolor="#979797">
<tr class="blue_1">
<td class="tdleft">运营商</td>
<td id="Frm_CarrierName" name="Frm_CarrierName" class="tdright">中国电信</td>
</tr>
<tr class="white_1">
<td class="tdleft">设备型号</td>
<td id="Frm_ModelName" name="Frm_ModelName" class="tdright">F460</td>
</tr>
<tr class="blue_1">
<td class="tdleft">设备标识号</td>
<td id="Frm_SerialNumber" name="Frm_SerialNumber" class="tdright">4C09B4-453004C
09B45B4692
</td>
</tr>
<tr class="white_1">
<td class="tdleft">硬件版本号</td>
<td id="Frm_HardwareVer" name="Frm_HardwareVer" class="tdright">V3.0</td>
</tr>
<tr class="blue_1">
<td class="tdleft">软件版本号</td>
<td id="Frm_SoftwareVer" name="Frm_SoftwareVer" class="tdright">V2.30.10P3T2hbH<
```

图 7-15　获取数据

好了，创建浏览器获取数据的过程已经运行了一遍了。下面可以使用 bs4 配合 Mechanize 来抓取光猫 F460 的数据了。

7.3 Mechanize 实站一：获取 Modem 信息

使用 urllib2 也可以比较方便地处理那些无须验证码的登录页面，不过使用 Mechanize 登录更加方便。当然是怎么方便怎么做。下面以抓取光猫 F460 的设置页面为例，使用 Mechanize 配合 Bs4 抓取光猫 F460 的数据。

7.3.1　获取 F460 数据

启动 Eclipse，新建 PyDev 项目 MechanizeAndBs4。在新项目中创建一个 PyDev Module 文件 getF460Info.py。

【示例 7-1】getF460Info.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016年8月25日

@author: hstking hstking@hotmail.com
```

```
'''

import mechanize
from bs4 import BeautifulSoup
from mylog import MyLog as mylog


class F460Info(object):
    '''获取光猫 f460 的信息 '''
    def __init__(self):
        self.url = 'http://192.168.1.1/'
        self.log = mylog()
        self.username = 'admin'
        self.password = '*******' #这里输入光猫的密码
        self.spider()


    def spider(self):
        responseContent = self.getResponseContent(self.url)
        if not responseContent:
            self.log.error('the response is null')
            exit()
        soup = BeautifulSoup(responseContent, 'lxml')
        modemInfo = {}
        modemInfo['CarrierName'] = soup.find('td', attrs={'id':'Frm_CarrierName'}).get_text().strip()
        modemInfo['modelName'] = soup.find('td', attrs={'id':'Frm_ModelName'}).get_text().strip()
        modemInfo['SerialNumber'] = soup.find('td', attrs={'id':'Frm_SerialNumber'}).get_text().strip()
        modemInfo['HardwareVer'] = soup.find('td', attrs={'id':'Frm_HardwareVer'}).get_text().strip()
        modemInfo['SoftwareVer'] = soup.find('td', attrs={'id':'Frm_SoftwareVer'}).get_text().strip()
        modemInfo['BootVer'] = soup.find('td', attrs={'id':'Frm_BootVer'}).get_text().strip()
        modemInfo['VerDate'] = soup.find('td', attrs={'id':'Frm_VerDate'}).get_text().strip()

        self.pipeline(modemInfo)


    def getResponseContent(self, url):
        self.log.info(u'begin create mechanize browser')
        br = mechanize.Browser()
        br.set_handle_equiv(True)
        br.set_handle_gzip(False)
```

```
        br.set_handle_redirect(True)
        br.set_handle_referer(True)
        br.set_handle_robots(False)
        br.set_handle_refresh(mechanize._http.HTTPRefreshProcessor(),
max_time=1)
    br.addheaders = [('User-agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36')]

        self.log.info(u'open url on mechanize browser')
        try:
            br.open(url)
        except:
            self.log.error(u'open %s failed' %url)
            return ''
        br.select_form(nr=0)
        br.form['Username'] = self.username
        br.form['Password'] = self.password
        br.submit()

        newUrl = url + 'template.gch'
        try:
            br.open(newUrl)
        except:
            self.log.error(u'open %s failed' %newUrl)
            return ''
        else:
            return br.response().read()


    def pipeline(self, info):
        fileName = u'f460ModemInfo.txt'.encode('gbk')
        with open(fileName, 'w') as fp:
            for key in info.keys():
                print('%s \t %s \n' %(key.encode('utf8'),
info.get(key).encode('utf8')))
                fp.write('%s \t %s \n' %(key.encode('utf8'),
info.get(key).encode('utf8')))


if __name__ == '__main__':
    fi = F460Info()
```

然后将 mylog.py 复制到 MechanizeAndBs4 项目下，单击 Eclipse 图标栏的运行图标，执行结果如图 7-16 所示。

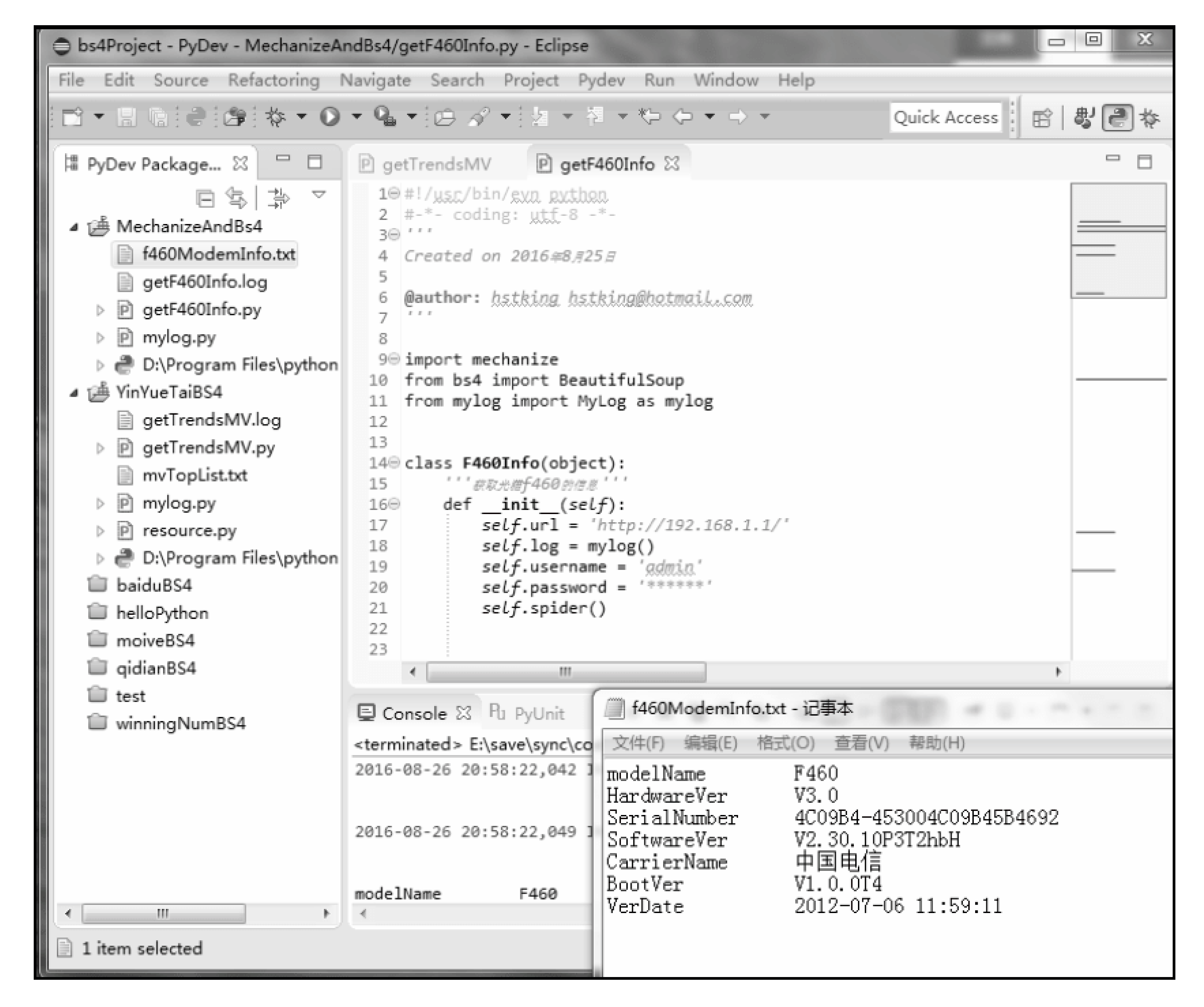


图 7-16　run getF460Info.py

爬虫程序运行无误，已经得到了预期的效果。

7.3.2　代码分析

这个爬虫大部分和以前的 bs4 爬虫没什么区别，只是用 Mechanize 模块代替了 urllib2 模块。在不需要输入验证码的情况下，Mechanize 还是很简单方便的。下面来看看示例 7-1 这个程序中的代码作用。

第 9~11 行是导入模块，很标准的 Python 程序流程。

第 16~21 行是 F460Info 类的解析函数，定义了几个变量。在 C 语言中定义这种类似的变量，一般都是在文件头使用 define。Python 中没有 define，放在这里正好合适，修改也很方便。

第 24~39 行是 spider 类函数。这个函数的作用是通过 BeautifulSoup 从字符串中过滤抓取所需的数据。在使用 soup 获取数据后，都使用了 strip 函数去除了数据左右的空格、回车等不可见字符。

第 42~71 行是 getResponseContent 类函数。作用是通过 Mechnize 模块来获取目标页面的返回数据。第 44 行创建了一个浏览器对象，第 45~51 行都是对浏览器对象的设置。这些设置并不是可有可无的，在打开某些网页时会因为这些设置的不同而得到不同的结果。如果没什么特殊要求，这样设置就可以了。

在编写程序之前，已经知道了最终的目标网页是 http://192.168.1.1/template.gch。可在第

53~62 行还是用浏览器对象打开了 http://192.168.1.1。这是因为直接打开目标页面是得不到任何数据的，只有先登录 http://192.168.1.1，得到合法的 Cookie，然后利用这个 Cookie 才能打开目标页面。

第 74~79 行的 pipeline 类函数的作用是处理最终的结果，将结果存入文件。这里直接使用 open 打开文件，数据中有中文字符，存入数据时必须使用 encode 将字符串转换成合法的数据后存入。

7.4 Mechanize 实战二：获取音悦台公告

上节讲的是无验证码登录爬取数据，这节演示需要验证码的爬虫了。有些网站或论坛为了防止暴力破解，在登录框设置了一个验证码。有坚固的盾就有锐利的矛，目前针对验证码的解决方案可谓是千奇百怪。有些方案的确有效，但不具备普遍性。考虑到爬虫所需要的只是数据，完全可以绕过验证码，直接使用 Cookie 登录就可以了。

7.4.1 登录原理

以音悦台网站为例，先来看看音乐台的登录界面，如图 7-17 所示。



图 7-17 音悦台登录界面

这个网站的登录就相当的麻烦了，它需要拖动滑块到合适的位置补全图片。如果只是验证码还有些开源方案可供选择。这种验证方式，目前还没发现可以模拟登录的 Python 程序。因此干脆选择适应性最广的方法，直接利用 Cookie 获取目标页面数据。

这种方法好处在于不管有没有验证码，也不管验证码有多么复杂，它都是有效的。它利用的只是 Cookie，跟用户名、密码、验证码都没有关系。缺点就是操作比较复杂，还有就是

Cookie 的生存期可能不长，过一段时间就得重新操作一遍。

7.4.2　获取 Cookie 的方法

获取 Cookie 的方法很多。不管使用哪种方法，首先都得登录后再操作。打开登录页面，输入用户名密码，将滑块拖动到正确的位置后登录网站，如图 7-18 所示。



图 7-18　登录网站

登录网站后进入目标页面，如图 7-19 所示。



图 7-19　目标页面

从目标页面可以获取个人的信件、站内公告等。现在只需要从目标界面获取 Cookie 就可以了。其他的数据留给 bs4 处理。

获取 Cookie 的方法很多，以下只列出比较典型的几种。

1．JavaScript 获取 Cookie

所有的浏览器默认情况下都是支持 JavaScript 的（如果默认不支持，请自行修改选项）。因此获取 Cookie 最常见到的方法就是在浏览器中打开目标页面，然后在地址栏输入 JavaScript 命令：

```
javascript:document.write(document.cookie)
```

执行结果如图 7-20 所示。



图 7-20　JavaScript 获取 Cookie

这种方法的好处在于无须借助任何工具就可以获取到 Cookie 信息，缺点是获取的 Cookie 信息有时会不太完整，缺少关键的几项。有的网站用这种方法获取的 Cookie 可以登录，有的又不行。不具备普遍性，这种方法不可取。

2．浏览器记录中获取 Cookie

浏览器在登录站点后会将 Cookie 信息保存到文件中（这里以 Chrome 浏览器为例）。这个文件的位置在 C:\Users\WindowsLoginName\AppData\Local\Google\Chrome\User Data\Default 目录，文件名为 Cookies，如图 7-21 所示。

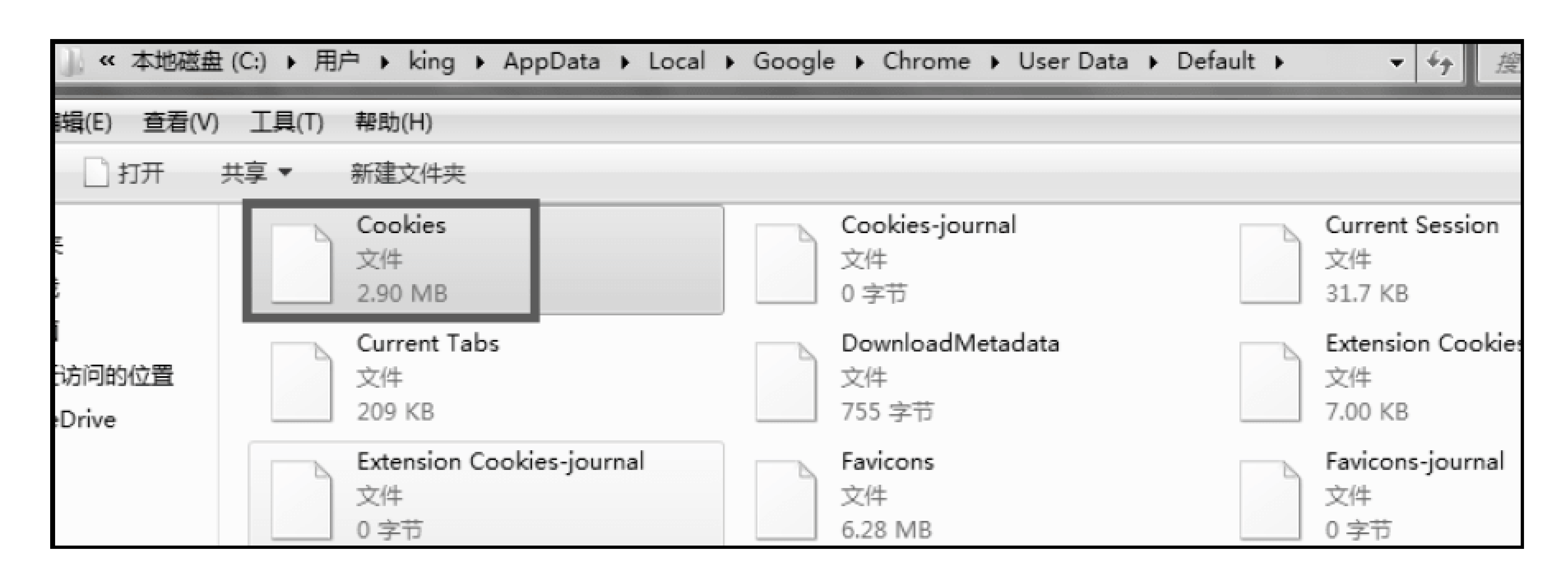


图 7-21　浏览器 Cookies 文件位置

这个 Cookies 文件实际上是一个 sqlite3 的数据库。Chrome 将浏览器上的所有 Cookie 都

保存到这个数据库中。将这个 Cookies 文件复制一个备份，名为 Cookies.db（尽量避免直接对系统文件操作）。在该目录下按 Shift 键并单击鼠标，在弹出的菜单中选取“在此处打开命令窗口”，如图 7-22 所示。

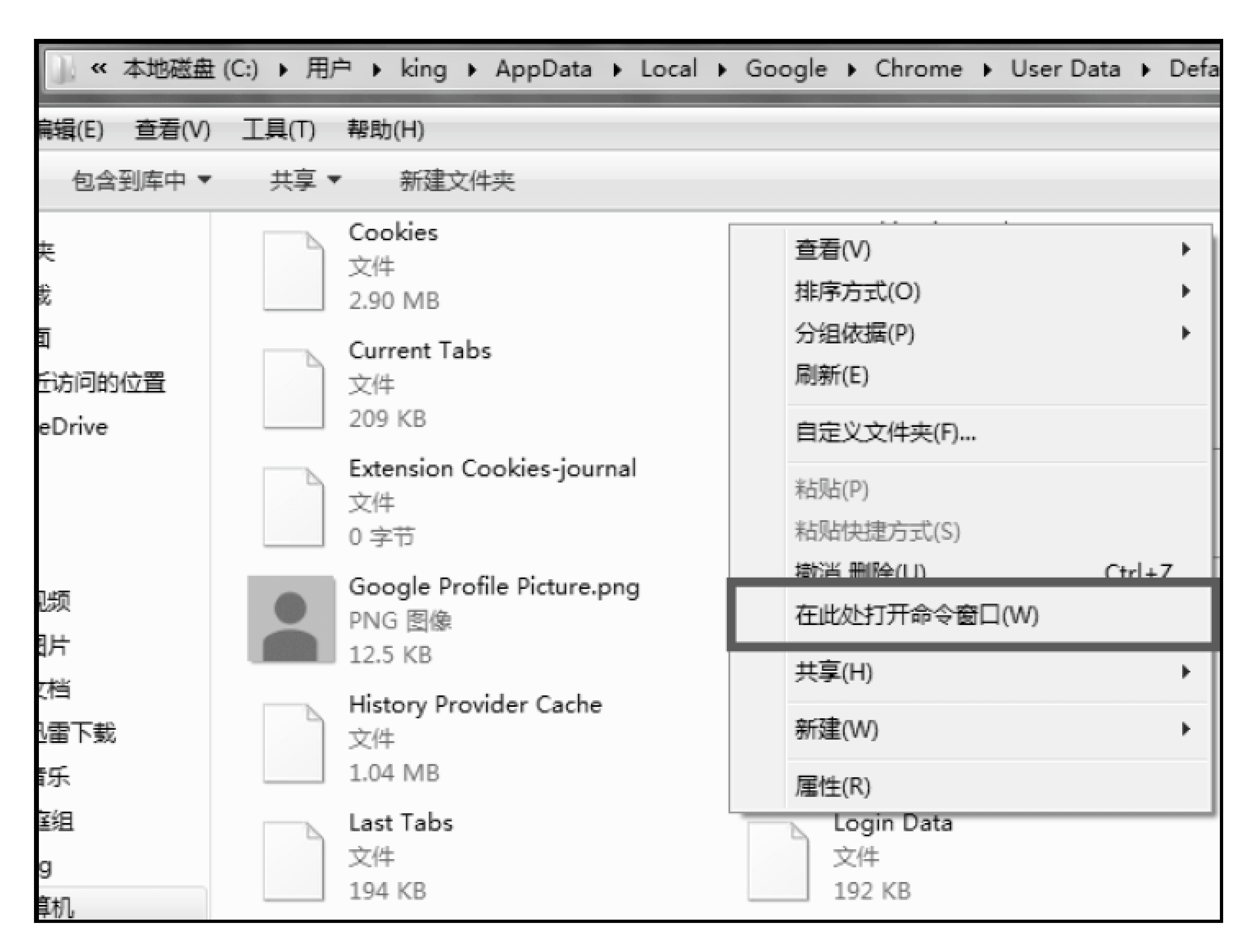


图 7-22 目录中打开 cmd.exe

在 cmd.exe 中打开 python 环境，连接到 sqlite3 数据库，并读出与 yinyuetai.com 相关的 Cookie。执行命令：

```
    Import sqlite3
    conn = sqlite3.connect('Cookies.db')
    for row in conn.execute('select * from Cookies where host_key like
"%yinyuetai.com%"'):
        print row
```



新版 Chrome 支持的 sqlite3 版本必须是 3.8 以上的,而默认安装的 Python 2.7 自带的版本是 3.6.21。所以需要到 sqlite3.org 上下载 Windows 版本的新的 sqlite3.dll 替换。如果不愿意替换，那就把 Cookie.db 文件复制上传到 Linux 处理。目前 Linux 自带的 Python 2.7 自带的版本是 3.8.7.1。

执行结果如图 7-23 所示。

```
6393.   '0100, 0, 0, 1, <read-write buffer ptr 0x7f0fbe5c56d8, size 262 at 0x7f0f
be5c5698>, 0)
(131      ,199  201, u'vchart.yinyuetai.com', u'CNZZ      ^^ .56', u'', u'/', : .
18  .     '000, 0, 0, 1311.     ^^354200, 1, 1, 1, <read-write buffer ptr 0x7f0fbe
4! `    , size 310 at 0x7f0fbe4b0498>, 0)
(1311'    .. )33: .J0, u'vchart.yinyuetai.com', u'CNZ          ^521', u'', u'/',    :
1855      ..000, 0, 0, 1311 '   ^^' 54200, 1, 1, 1, <read-write buffer ptr 0x7f0fbe
5b(    , size 326 at 0x7f0fbe5b6960>, 0)
(1311      27(    1, u'www.yinyuetai.com', u'CNZZ_      ^.456', u'', u'/', 1:    ^0
4912      )0, 0, 0, 1311 '        ^4100, 1, 1, 1, <read-write buffer ptr 0x7f0fbe5dc
c88, size 278 at 0x7f0fbe5dcc48>, 0)
(13^      : 85.   )0, u'so.yinyuetai.com', u'CNZ        30456', u'', u'/', 13_     4
91(    )00, 0, 0, 1311(    ^^^^73100, 1, 1, 1, <read-write buffer ptr 0x7f0fbe4b04
d8, size 310 at 0x7f0fbe4b0498>, 0)
(1311'      . 88   .J0, u'.yinyuetai.com', u'searchSID', u'', u'/', 0, 0, 0, 1:    `
040'   J100, 0, 0, 1, <read-write buffer ptr 0x7f0fbe5c56d8, size 262 at 0x7f0fbe
5c5698>, 0)
(1311         .8^ )0, u'www.yinyuetai.com', u'route', u'', u'/', 0, 0, 0, 13      ^
410'   100, 0, 0, 1, <read-write buffer ptr 0x7f0fbe5c5448, size 262 at 0x7f0fbe5
c5408>, 0)
(1311 ^   . J02(    )0, u'v.yinyuetai.com', u'CNZZ^      J0456', u'', u'/', 13      49
90(  ^00, 0, 0, 13116           4100, 1, 1, 1, <read-write buffer ptr 0x7f0fbe4b04d
8, size 310 at 0x7f0fbe4b0498>, 0)
>>>
```

图 7-23　从文件中获取 Cookie

已经将所有相关的 Cookie 列出来了。如果要把这些数据转换成可使用 Cookie，还得继续将其中的 encrypted_value 字段解码。这个是可以做到，得安装别的 Python 模块，相当不方便。使用这种方法获取 Cookie，好处是所有的 Cookie 内容都一网打尽，连用户名密码都可以用明文解读出来；坏处则是要把这些数据转换成 Mechanize 可用的 Cookie 比较麻烦，还需要安装其他的第三方模块，有些鸡肋。

3．利用工具获取 Cookie

最后的方法就是利用网络工具，在浏览器向服务器发送数据时截取这些数据，这些数据不仅仅包括 Cookie，还有一些其他的信息，而且这些信息 Mechanize 还都用得上，简直是完美。这种方法与 Mechanize 相当的合拍，都是往服务器发送数据，区别仅在于一个是浏览器发送，一个是 Mechanize 模块发送而已。

截取浏览器和服务器之间的网络工具有很多，比如 Fiddler、Wireshark 和 BurpSuite，也有浏览器自带的，比如 Firefox 的 Httpfox 和 Chrome 开发工具。个人建议是直接使用 Chrome 的开发工具，如果 Chrome 开发工具截取的数据不能使用（这种可能性极低）或者没使用 Chrome 浏览器，那就使用 Fiddler 或 BurpSuite。

7.4.3　获取 Cookie

1．Chrome 开发工具获取 Cookie

Chrome 浏览器自带的开发功能相当强大，这里只使用它的抓包功能。在浏览器中打开目标网站并登录，进入目标页面，按 F12 键，打开 Chrome 开发工具，如图 7-24 所示。



图 7-24　Chrome 开发工具

在 Chrome 浏览器下方的开发工具中单击 Network 标签页。按 F5 键，刷新页面。会在浏览器中得到很多数据，然后在 Filter 框中输入目标页面的关键词 bulletin，找到发送请求的 Request，如图 7-25 所示。

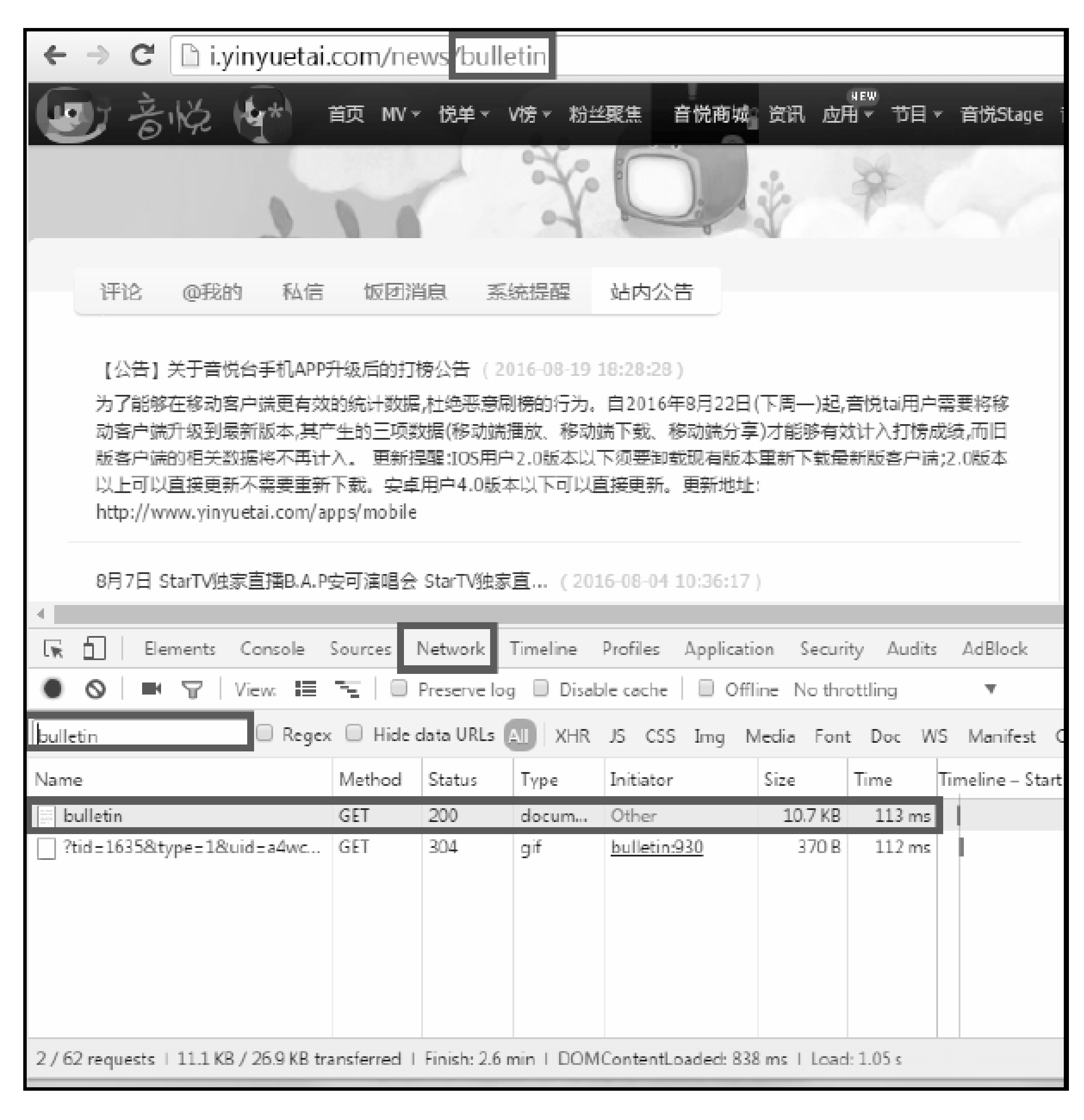


图 7-25　find Request

单击这个 Name 为 bulletin 的 Request，在打开的界面中单击 Headers 标签，得到这个 Reqeust 的 Headers（这里也有 Cookies 标签，但它的表现形式是表格，另外所需的数据不只是 Cookie，还有 User-Agent，所以这里选择 Headers 标签），如图 7-26 所示。

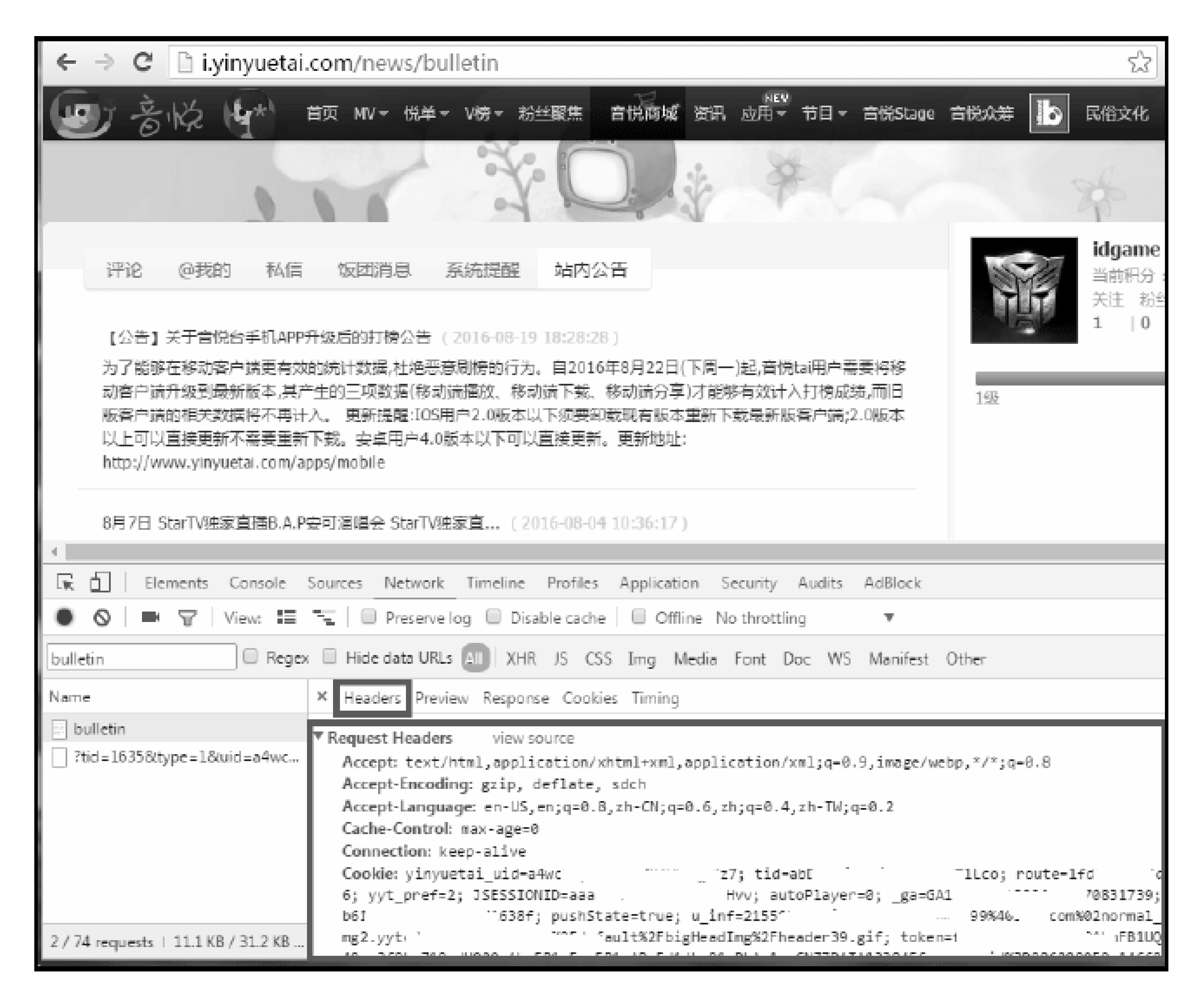


图 7-26　headers

将这个 Request Headers 里的所有数据都复制到一个文本文件 headersRaw.txt 中备用。

2．BurpSuite 获取 Cookie

如果不喜欢 Chrome 浏览器的开发工具，或者没有使用 Chrome 浏览器，那也可以使用工具来获取 Cookie。这里笔者选择的是 BurpSuite 工具，BrupSuite 工具简单方便，跨平台运行，功能强大。如果要说明 BurpSuite 的其他功能，恐怕得一本厚书才行。这里只使用 BrupSuite 最简单的抓包功能。打开 BurpSuite 工具，选择 Proxy 标签下的 Intercept 标签，将 Intercept is on 按钮激活。这样设置将截取浏览器的 Request，但不向服务器发送，如图 7-27 所示。

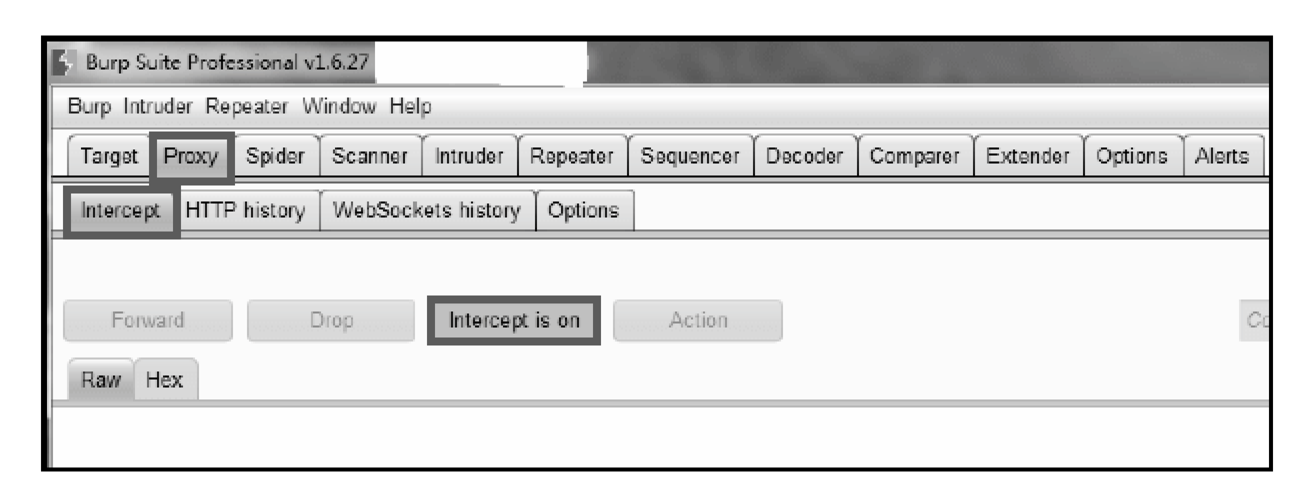


图 7-27　BurpSuite

BurpSuite 监控的端口是本机的 8080 端口，所以必须将浏览器的代理端口设置为 127.0.0.1:8080。这个设置根据选择的浏览器不同而选择不同的方法设置。如果使用的是 Chrome，建议使用 SwitchySharp 插件。如果使用的是 FireFox，建议使用 FoxyProxy。至于其

他的浏览器，就干脆将系统代理设置成 127.0.0.1，然后将所有浏览器设置成使用系统代理好了。

打来浏览器，设置好代理，然后刷新登录后的目标网页。BurpSuite 将得到数据，如图 7-28 所示。

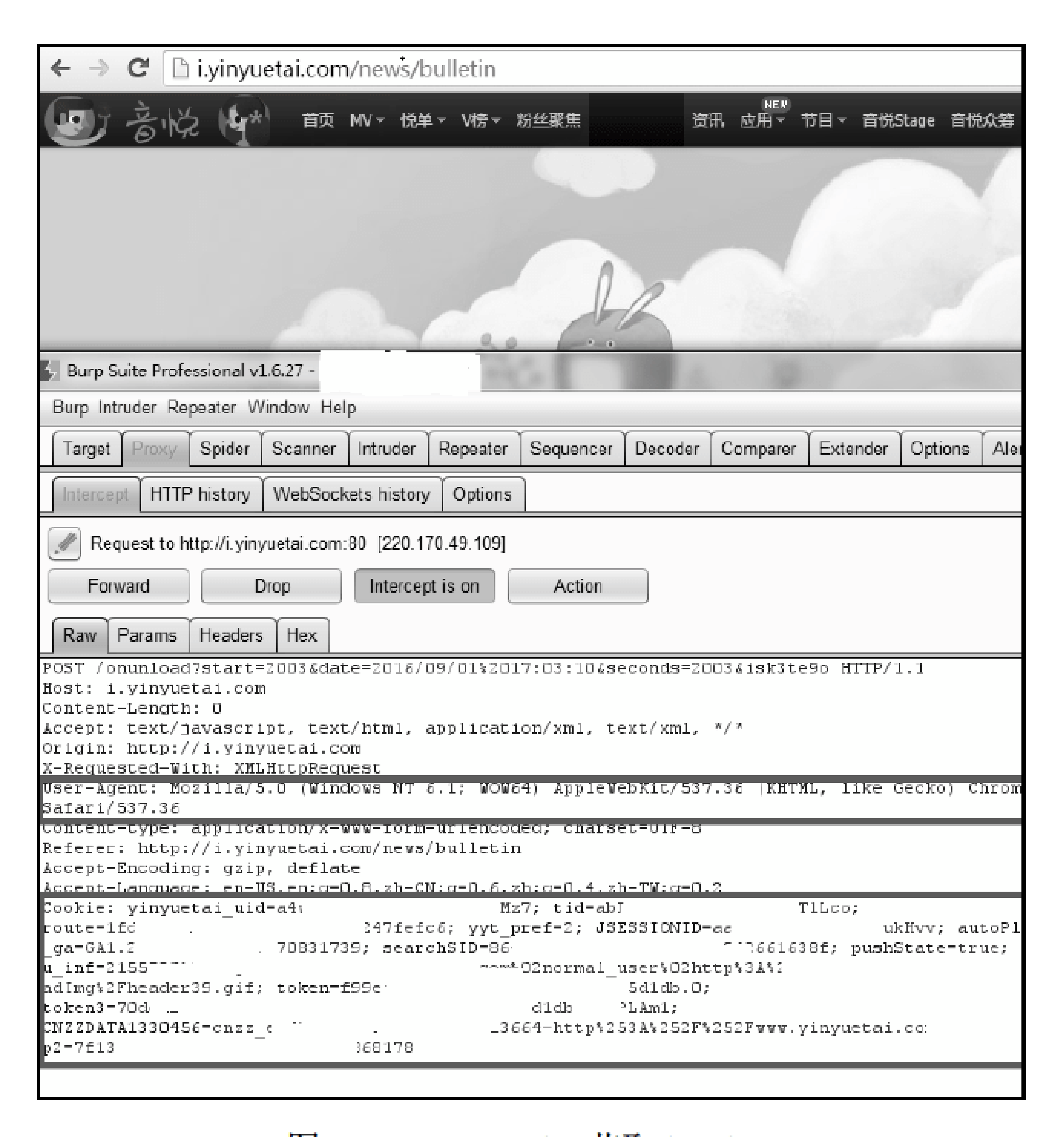


图 7-28　BurpSuite 获取 headers

主要是获取 Cookie 和 User-Agent 的数据。将这个 Raw 标签内的所有数据复制到文本文件 headersRaw.txt 中备用。

这两种获取 headersRaw.txt 文件的方法任选一种都可以，然后为它写一个程序，将所需的数据按照所需的格式导出来。打开 Eclipse，创建 PyDev Project 项目 getBulletin，将 headersRaw.txt 文件复制到 getBulletin 项目下，并在项目下创建一个名为 getHeadersFromFile 的 pyDev Modules。

【示例 7-2】getHeaders.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016 年 9 月 1 日

@author: hstking hstking@hotmail.com
'''

```

```

def getHeaders(fileName):
    headers = []
    headerList = ['User-Agent', 'Cookie']
    with open(fileName, 'r') as fp:
        for line in fp.readlines():
            name, value = line.split(':', 1)
            if name in headerList:
                headers.append((name.strip(), value.strip()))
    return headers

if __name__ == '__main__':
    headers = getHeaders('headersRaw.txt')
    print headers
```

测试 getHeadersFromeFile.py，单击 Eclipse 图标栏的运行图标，执行结果如图 7-29 所示。

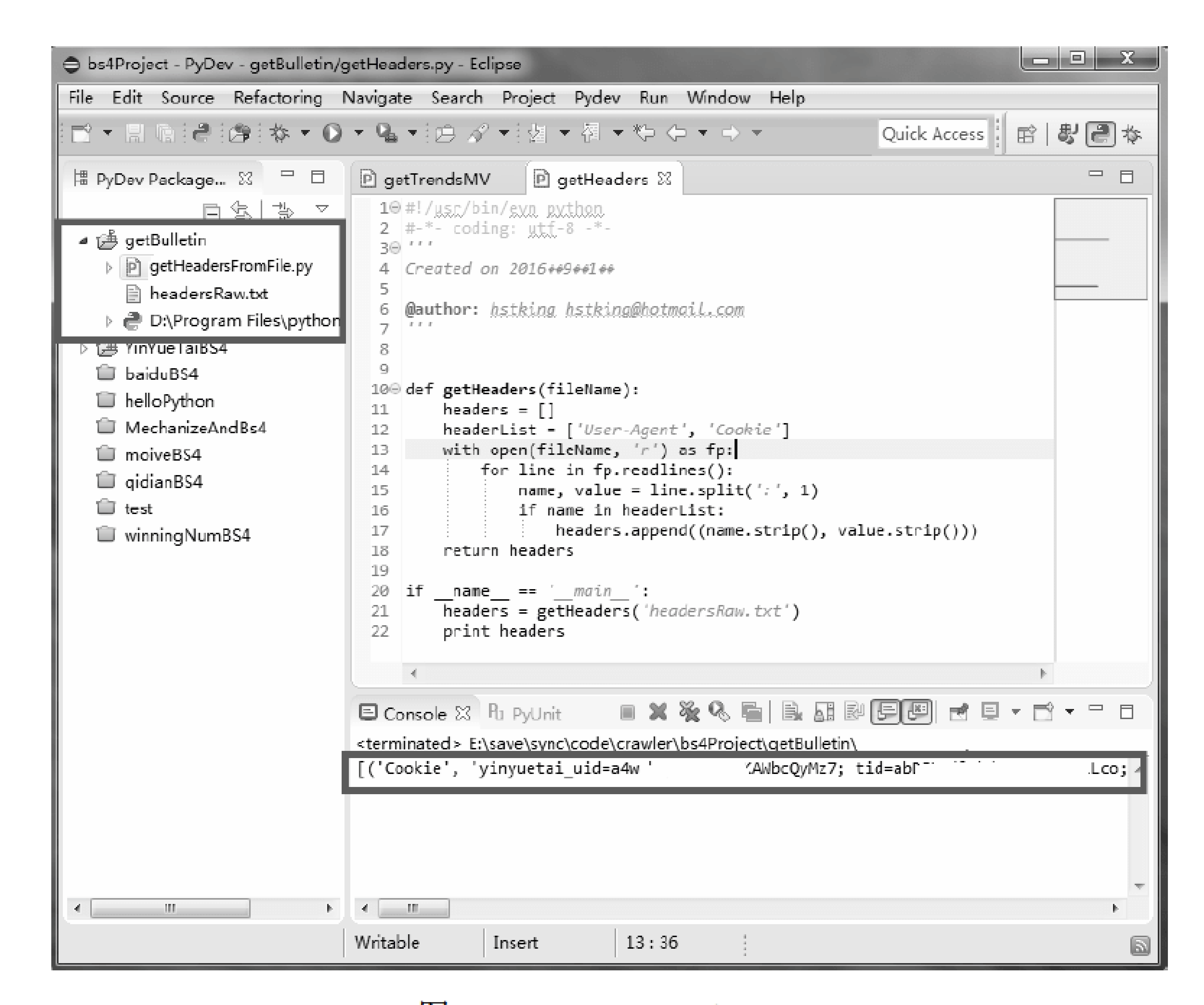


图 7-29 run getHeaders.py

已经将 Cookie 和 User-Agent 过滤出来并按照格式排列好了，最后所得到的 headers 是一个包含了 2 个元组的列表。

7.4.4 使用 Cookie 登录获取数据

获取音悦台网站个人站内公告的充分条件已经具备了。下面开始使用 Mechanize 和 bs4

来获取个人公告数据。

打开 Eclipse，进入刚建立的 getBulletin 项目中，将以前项目中使用的 mylog.py 复制到 getBulletin 项目下，并在项目中创建一个新的 PyDev Module，文件名为 getYinyuetaiBulletin.py。

【示例 7-3】getYinyuetaiBulletin.py 的内容如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016 年 9 月 1 日

@author: hstking hstking@hotmail.com
'''

import mechanize
from bs4 import BeautifulSoup
from mylog import MyLog as mylog
from getHeadersFromFile import getHeaders
import codecs

class Item(object):
    title = None
    content = None


class GetBulletin(object):
    def __init__(self):
        self.url = 'http://i.yinyuetai.com/news/bulletin'
        self.log = mylog()
        self.headersFile = 'headersRaw.txt'
        self.outFile = 'bulletin.txt'

        self.spider()


    def getResponseContent(self, url):
        self.log.info('begin use mechanize module get response')
        br = mechanize.Browser()
        br.set_handle_equiv(True)
        br.set_handle_redirect(True)
        br.set_handle_referer(True)
        br.set_handle_robots(False)
```

```

        br.set_handle_refresh(mechanize._http.HTTPRefreshProcessor(), max_time=1)
        headers = getHeaders(self.headersFile)
        br.addheaders = headers
        br.open(url)
        return br.response().read()


    def spider(self):
        self.log.info('beging run spider module')
        items = []
        responseContent = self.getResponseContent(self.url)
        soup = BeautifulSoup(responseContent, 'lxml')
        tags = soup.find_all('div', attrs={'class':'item_info'})
        for tag in tags:
            item = Item()
            item.title = tag.find('p', attrs={'class':'title'}).get_text().strip()
            item.content = tag.find('p', attrs={'class':'content'}).get_text().strip()
            items.append(item)
        self.pipelines(items)


    def pipelines(self, items):
        self.log.info('begin run pipeline function')
        with codecs.open(self.outFile, 'w', 'utf8') as fp:
            for item in items:
                fp.write(item.title + '\r\n')
                self.log.info(item.title)
                fp.write(item.content + '\r\n')
                self.log.info(item.content)
                fp.write('\r\n'*8)


if __name__ == '__main__':
    GB = GetBulletin()
```

单击 Eclipse 图标栏的运行图标，执行结果如图 7-30 所示。

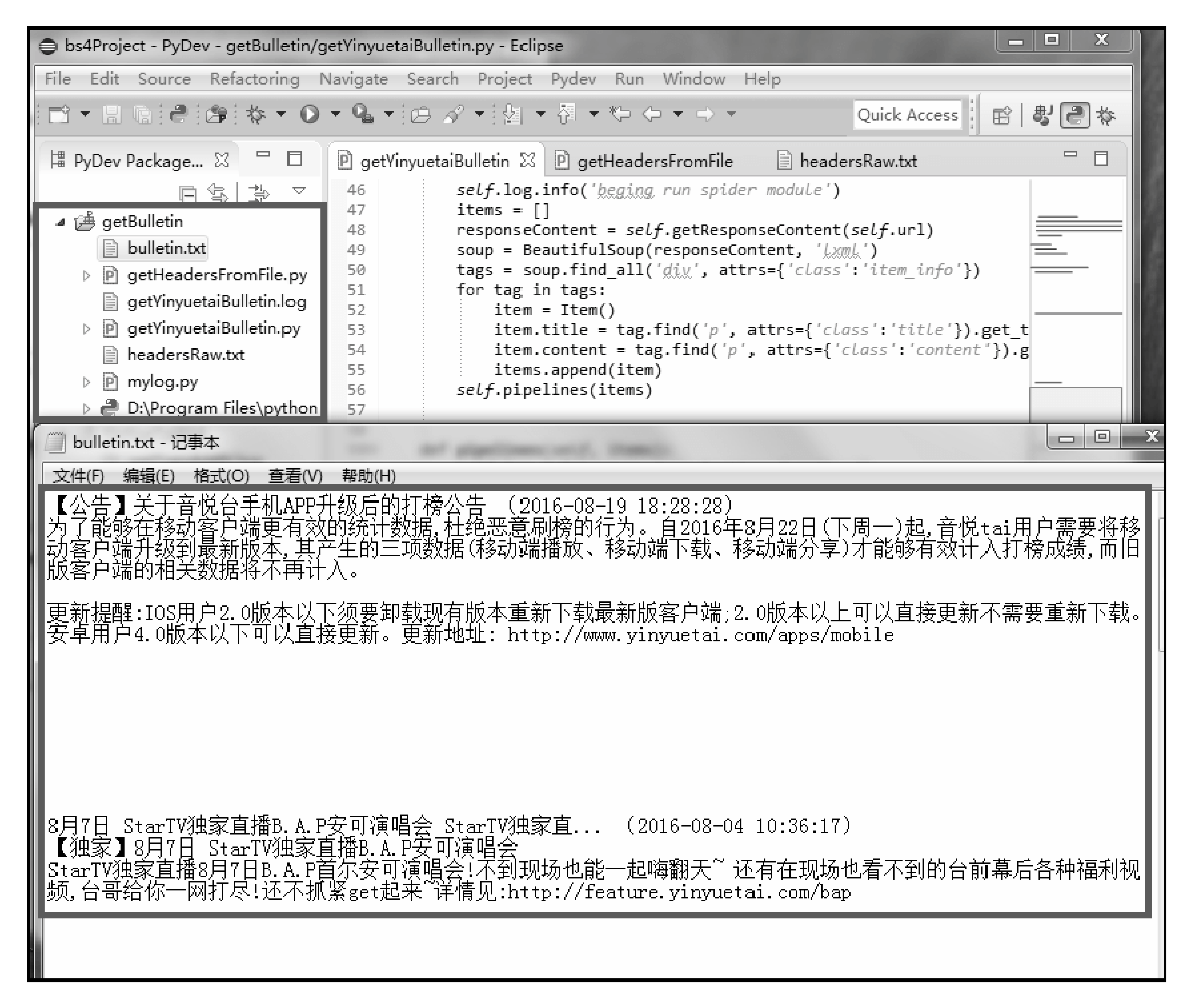


图 7-30　爬虫抓取的公告

已经成功地获取了音悦台的个人公告，如图 7-31 所示。



图 7-31　目标网页上的公告

与网站上的个人公告比较一下，完全一样，爬虫没有问题。Mechanize 使用 Cookie 登录，除了 Cookie 的生存期问题，算是个非常好的办法，要比 urllib2 模块模拟浏览器要方便得多。

7.5 本章总结

Mechanize 不是爬虫，它不是得到爬虫结果的充要条件，但在某些时候比爬虫更加重要。毕竟爬虫过滤的来源数据要靠 Mechanize 来获取。大多数时候的确可以用别的模块来替代 Mechanize，这样一来过程就未免有些复杂了。虽然爬虫程序追求的只是结果，过程是否繁杂对结果没有影响，但能用简单的模块解决问题就没必要用复杂的方法。

第 8 章

Selenium模拟浏览器

Python 网络爬虫中最麻烦的不是那些需要登录才能获取数据的网站，而是那些通过 JavaScript 获取数据的站点。Python 对 JavaScript 的支持不太好。想用 Python 获取网站中 JavaScript 返回的数据，唯一的方法就是模拟浏览器了。这个模拟浏览器跟 Mechanize 模块稍有不同，Mechanize 模块并不支持 JavaScript，所以这里需要一款可以模拟真实浏览器的模块——Selenium 模块。

8.1 安装 Selenium 模块

Selenium 是一套完整的 Web 应用程序测试系统，它包含了测试的录制（Selenium IDE）、编写及运行（Selenium Remote Control）和测试的并行处理（Selenium Grid）。Selenium 的核心 Selenium Core 基于 JsUnit，完全由 JavaScript 编写，因此可运行于任何支持 JavaScript 的浏览器上。

8.1.1 Windows 下安装 Selenium 模块

在 Windows 中安装 Selenium 模块还是采用最简单的 pip 安装，执行命令：

```
pip install selenium
```

执行结果如图 8-1 所示。

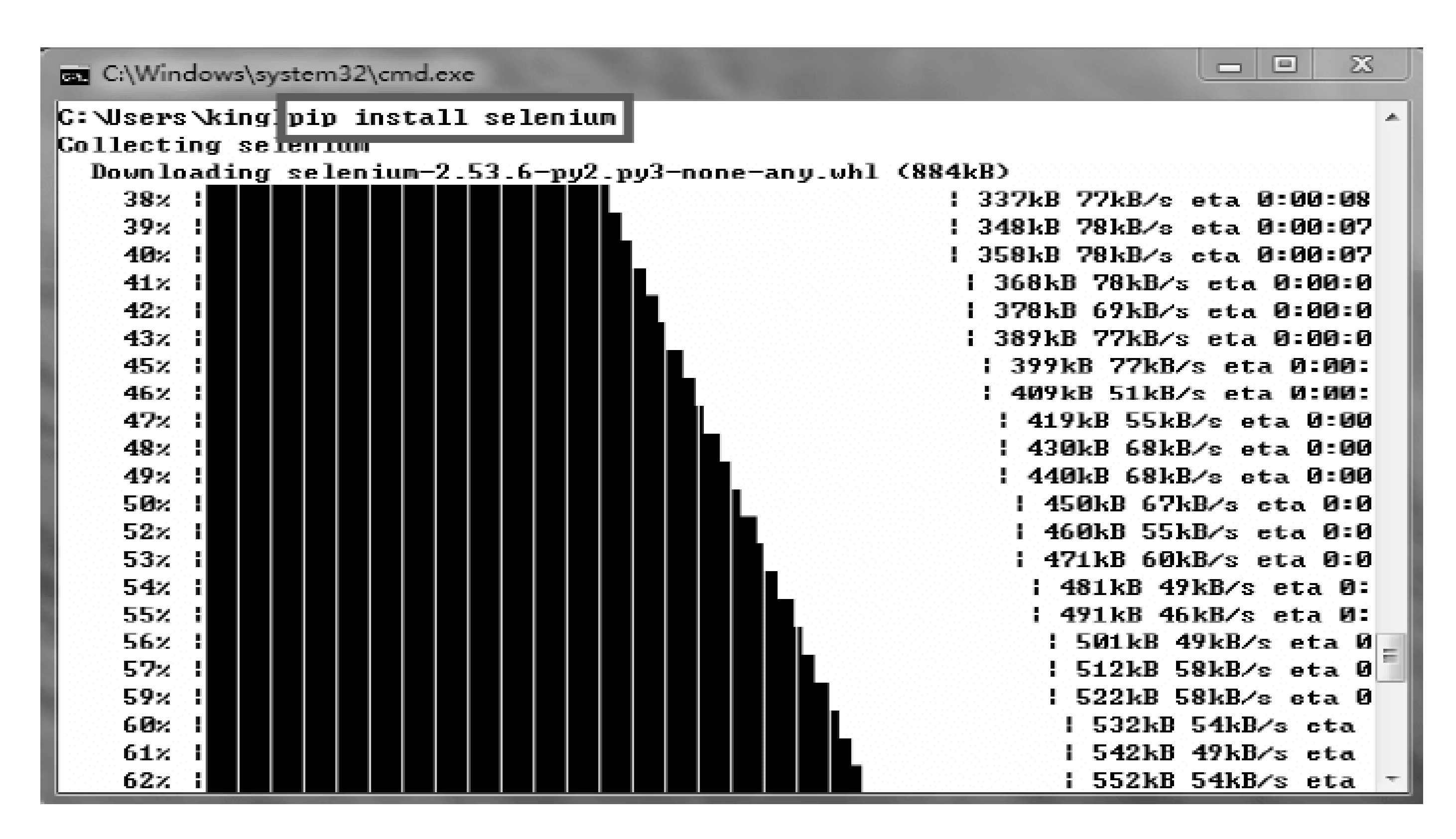


图 8-1　Windows 安装 Selenium

Windows 中安装 Selenium 完毕，可以直接使用了。

8.1.2 Linux 下安装 Selenium 模块

在 Linux 中安装软件尽可能地使用 apt-get，这样便于管理软件。执行命令：

```
apt-get install python-selenium
```

执行结果如图 8-2 所示。

```
king@debian: ~
Last login: Wed Sep  7 01:00:57 2016 from 192.168.2.99
king@debian:~$ su
密码：
root@debian:/home/king# apt-get install python-selenium
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
建议安装的软件包：
  chromedriver firefoxdriver
下列【新】软件包将被安装：
  python-selenium
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 0 个软件包未被升级
。
需要下载 74.1 kB 的软件包。
解压缩后会消耗掉 497 kB 的额外空间。
获取：1 http://mirrors.163.com/debian/ jessie-backports/main python-selenium all
 2.48.0+dfsg1-2~bpo8+1 [74.1 kB]
下载 74.1 kB，耗时 0秒 (351 kB/s)
正在选中未选择的软件包 python-selenium。
(正在读取数据库 ... 系统当前共安装有 94130 个文件和目录。)
正准备解包 .../python-selenium_2.48.0+dfsg1-2~bpo8+1_all.deb  ...
正在解包 python-selenium (2.48.0+dfsg1-2~bpo8+1) ...
正在设置 python-selenium (2.48.0+dfsg1-2~bpo8+1) ...
root@debian:/home/king# exit
```

图 8-2　Linux 安装 Selenium

Linux 中安装 Selenium 完毕。

8.2 浏览器选择

在编写 Python 网络爬虫时，主要用到 Selenium 的 Webdriver。Selenium.Webdrive 不可能支持所有的浏览器，也没必要支持所有的浏览器。实际上目前流行的浏览器核心也就是那么几种。先看看 Selenium.Webdriver 支持哪几种浏览器。

8.2.1 Webdriver 支持列表

查看模块的功能，最简单也是最方便的方法就是直接使用 help 命令。打开 cmd.exe 工具，执行命令：

```
python
from selenium import webdriver
help(webdriver)
```

执行结果如图 8-3 所示。

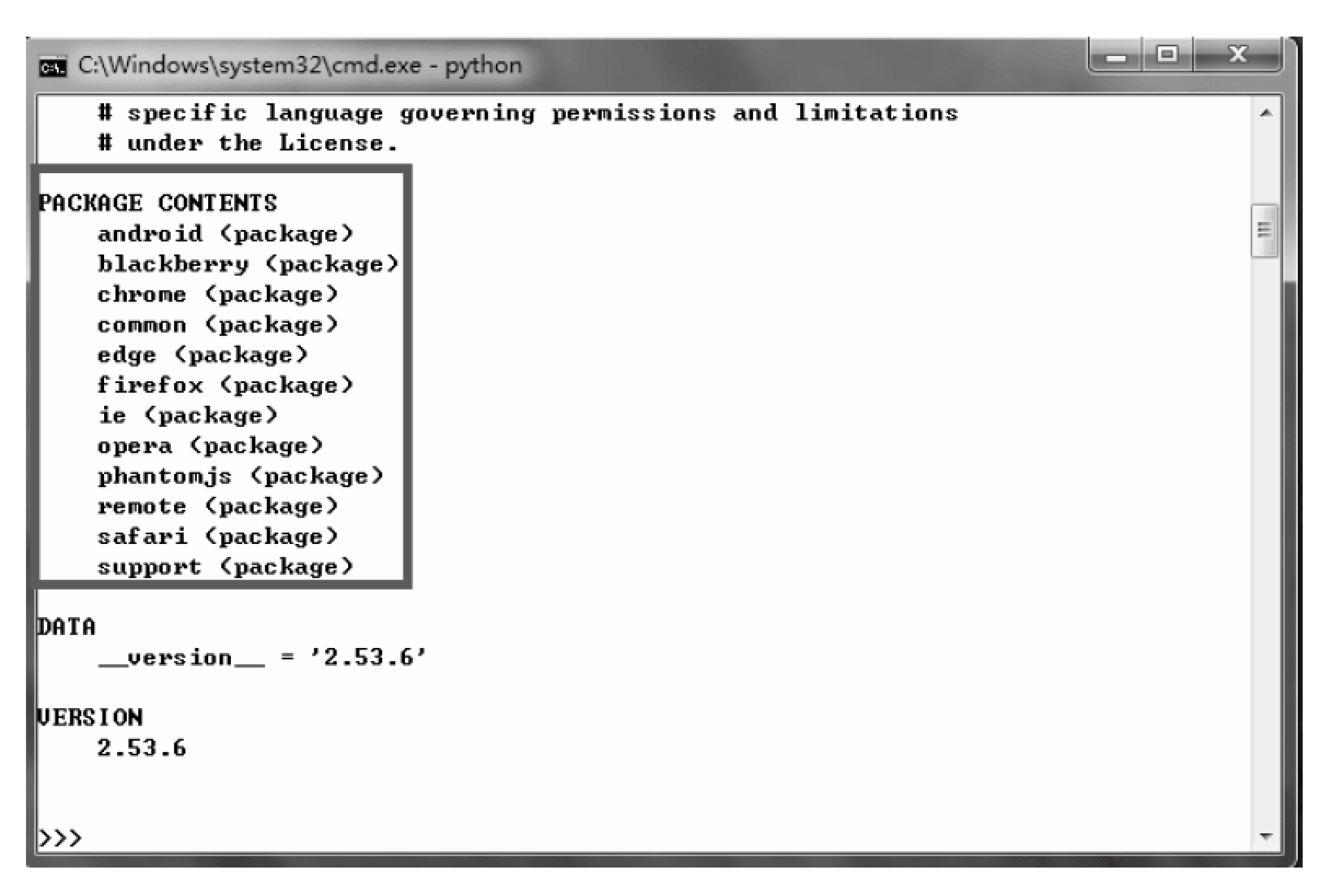


图 8-3　webdriver 支持列表

在以上列表中，android 和 blackberry 是移动端的浏览器，可以先去掉。移动端的浏览器虽然也支持 JavaScript，但跟 PC 端浏览器根本是两回事。Common 和 support 也可以先去掉，剩下的只有 Chrome、Edge、Firefox、IE、Opera、Phantomjs 和 Safari 了。Chrome、Dege、Firefox、IE、Opera、Safari 比较常见，而 PhantomJS 则有些名不见经传。

PhantomJS 是一个基于 WebKit 的服务器端 JavaScript API。它全面支持 Web 而不需浏览器支持，其快速、原生支持各种 Web 标准：DOM 处理、CSS 选择器、JSON、Canvas 和 SVG。 PhantomJS 可以用于页面自动化、网络监测、网页截屏，以及无界面测试等。

无界面，则意味着开销小，也意味着速度快。网上有牛人测试过，使用 Selenium 调用上面的浏览器，速度前三甲分别是 PhantomJS、Chrome 和 IE（remote 调用 HtmlUnit 速度才是最快的，但 HtmlUnit 对 JavaScript 的支持不太好），开销小、速度快对 JavaScript 的支持也不错。唯一的缺点是没有 GUI，但在服务器下运行程序时，这又成了优点。所以无须犹豫，就选 PhantomJS 了。事实上，在爬行 JavaScript 才能返回数据的网站时，没有比 Selenium 和 PhantomJS 更适合的组合了。

8.2.2　Windows 下安装 PhantomJS

PhantomJS 的官网主页是 http://phantomjs.org/。在浏览器中打开主页，单击 Download V2.1 按钮进入下载页面，如图 8-4 所示。

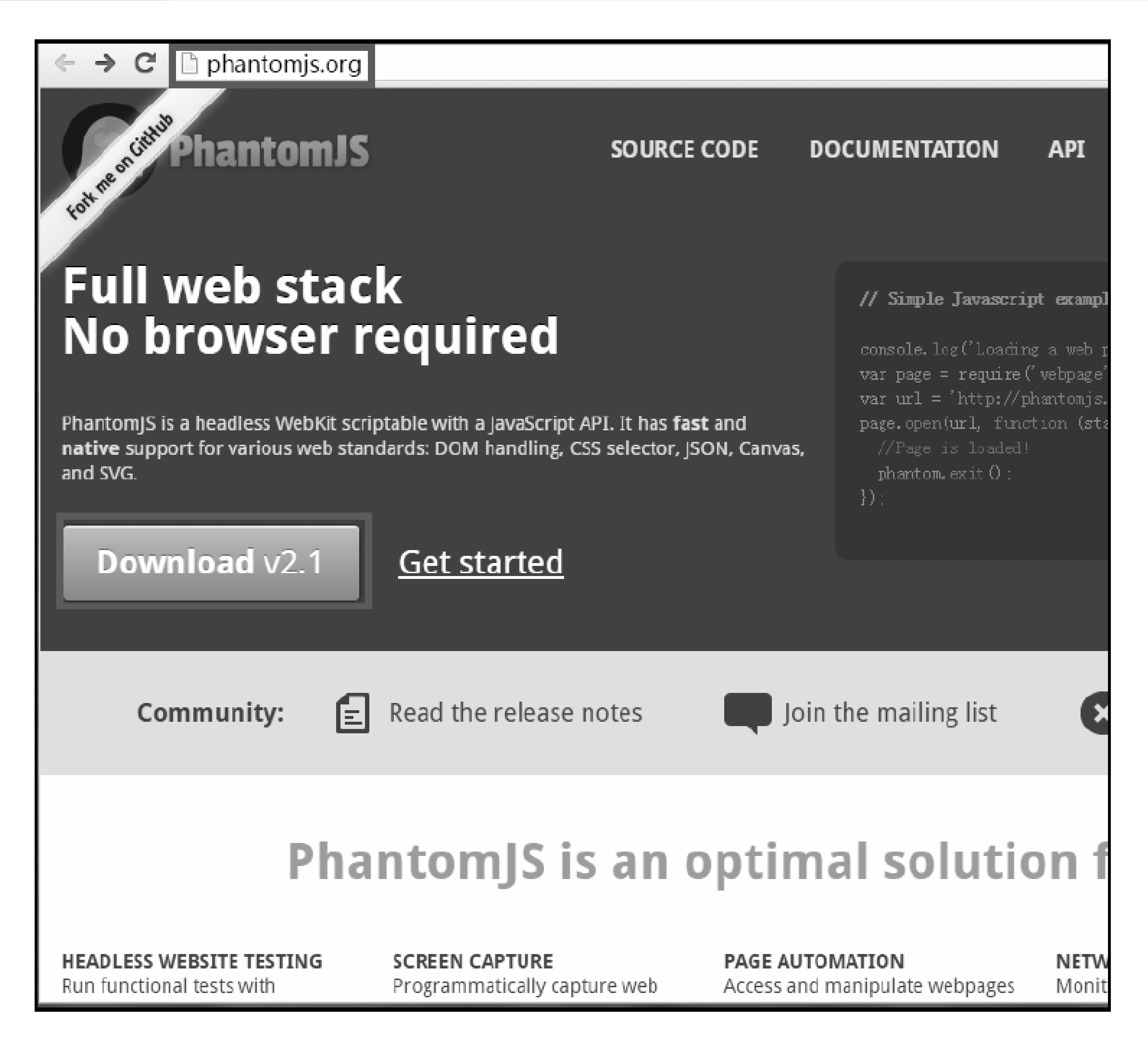


图 8-4　PhantomJS 官网主页

进入下载页面后，选择 Windows 版本的 PhantomJS 下载软件，如图 8-5 所示。

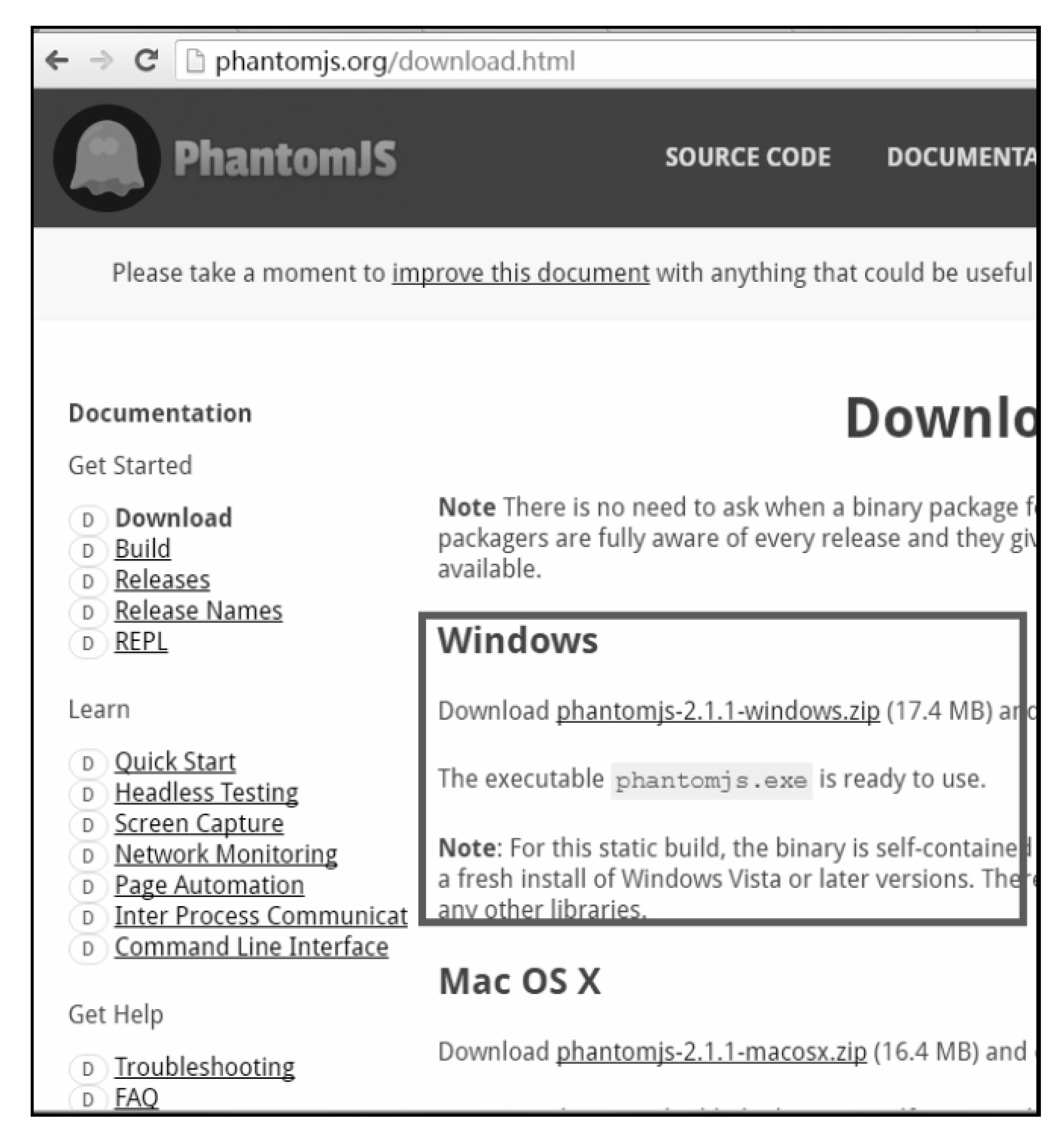


图 8-5　下载 Windows 版本 PhantomJS

因为未知的原因，直接用浏览器下载 PhantomJS 速度极慢。有时根本就没反应，建议使用迅雷下载 PhantomJS。迅雷上有用户曾下载过 PhantomJS 后，后面的迅雷用户再次下载速度就很快了。

下载完成后，解压压缩包，然后将 exe 文件加入到系统路径中就可以了。重新设置系统路径是很麻烦的事情。还记得安装 Python 2.7 的过程吗？安装程序已自动将 Python 2.7 的路径加入到了系统路径中了，反正 PhantomJS 也是配合 Python 使用的，直接将解压后的 PhtomJS.exe 复制到 Python 2.7 的目录中就可以了，如图 8-6 所示。



图 8-6　Windows 设置 PhantomJS 环境

在 Python 环境中测试一下，如图 8-7 所示。

```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\king>python
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from selenium import webdriver
>>> driver = webdriver.PhantomJS()
>>>
```

图 8-7　Windows 中测试 PhantomJS 环境

Windows 下的 PhantomJS 环境已配置好，可以直接使用了。

8.2.3　Linux 下安装 PhantomJS

还是打开 PhantomJS 官网的下载页面，选择合适的版本，使用迅雷下载，如图 8-8 所示。

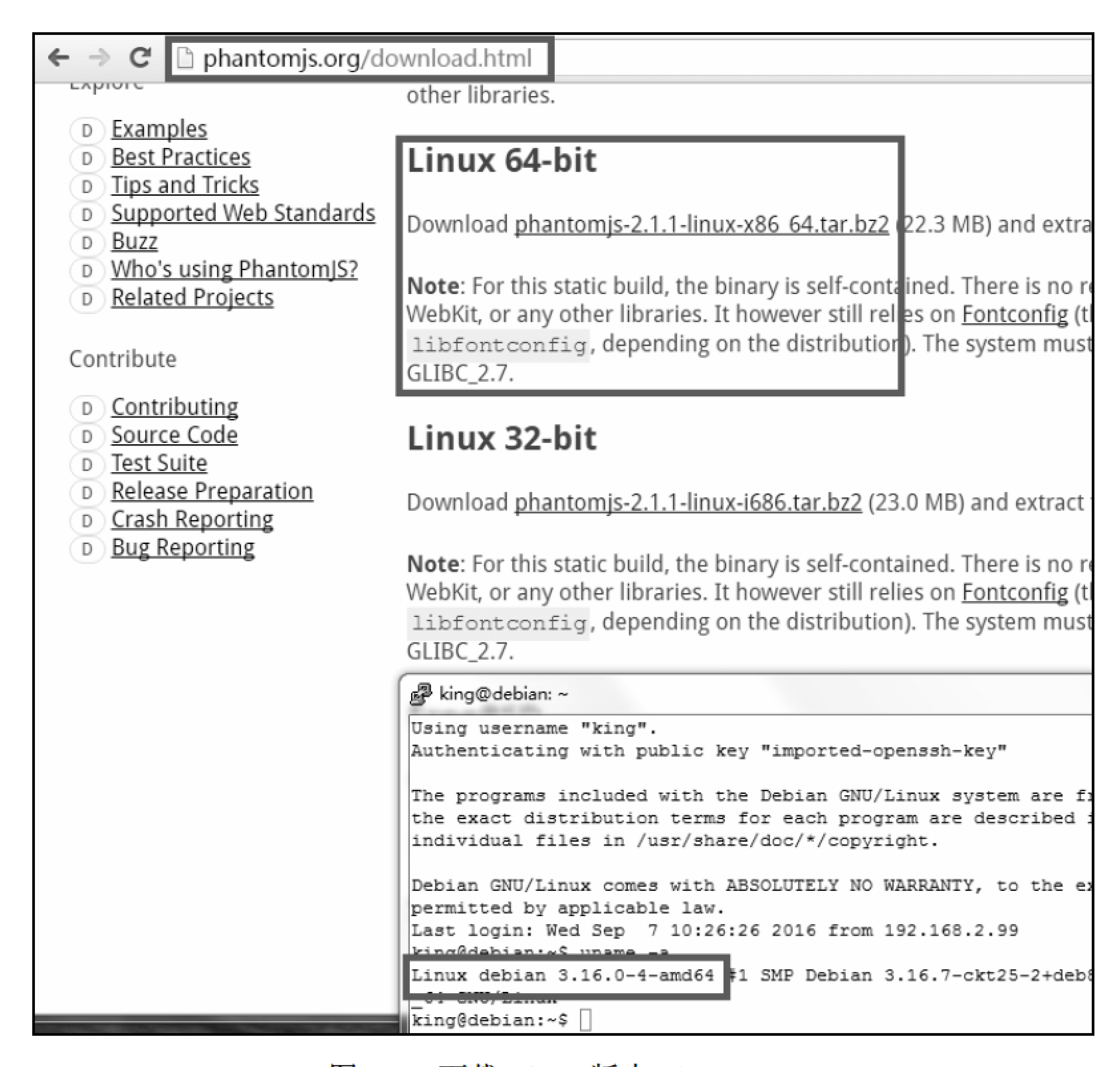


图 8-8　下载 Linux 版本 PhantomJS

将下载好的压缩文件上传到 Linux 后解压缩，然后将可执行文件复制到系统路径 /usr/local/bin 文件夹下（Linux 的系统路径有很多，随意选一个都可以）。打开 Putty，连接到 Linux 上，执行命令：

```
tar jxvf phantomjs-2.1.1-linux-x86_64.tar.bz2
cp phantomjs-2.1.1-linux-x86_64/bin/phantomjs /usr/local/bin/
```

```
ls -l /usr/local/bin/
```

执行结果如图 8-9 所示。

```
king@debian: /tmp
phantomjs-2.1.1-linux-x86_64/examples/features.js
phantomjs-2.1.1-linux-x86_64/examples/netsniff.js
phantomjs-2.1.1-linux-x86_64/examples/walk_through_frames.js
phantomjs-2.1.1-linux-x86_64/examples/printheaderfooter.js
phantomjs-2.1.1-linux-x86_64/examples/responsive-screenshot.js
phantomjs-2.1.1-linux-x86_64/examples/countdown.js
phantomjs-2.1.1-linux-x86_64/examples/detectsniff.js
phantomjs-2.1.1-linux-x86_64/examples/simpleserver.js
phantomjs-2.1.1-linux-x86_64/examples/postjson.js
phantomjs-2.1.1-linux-x86_64/examples/run-jasmine2.js
phantomjs-2.1.1-linux-x86_64/examples/run-jasmine.js
phantomjs-2.1.1-linux-x86_64/README.md
phantomjs-2.1.1-linux-x86_64/LICENSE.BSD
phantomjs-2.1.1-linux-x86_64/bin/
phantomjs-2.1.1-linux-x86_64/bin/phantomjs
phantomjs-2.1.1-linux-x86_64/third-party.txt
phantomjs-2.1.1-linux-x86_64/ChangeLog
king@debian:/tmp$ su
密码:
root@debian:/tmp# cp phantomjs-2.1.1-linux-x86_64/bin/phantomjs /usr/local/bin/
root@debian:/tmp# ls -l /usr/local/bin/
总用量 66344
-rwxr-xr-x 1 root staff 67932064 9月   7 10:46 phantomjs
root@debian:/tmp#
```

图 8-9　Linux 中设置 PhantomJS 环境

在 Python 环境中测试一下，如图 8-10 所示。

```
king@debian:/tmp$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from selenium import webdriver
>>> driver = webdriver.PhantomJS()
>>>
```

图 8-10　Linux 中测试 PhantomJS 环境

Linux 下的 PhantomJS 环境已配置好，可以直接使用了。

8.3 Selenium&PhantomJS 抓取数据

Selenium 和 PhantomJS 配合，可以模拟浏览器获取包括 JavaScript 的数据。问题是本文是讲爬虫的，现在不单要获取网站数据，还需要过滤出“有效数据”才行。这里就不用麻烦 bs4 了（实际上非要用 bs4 也不是不可以），Selenium 本身就带有一套自己的定位过滤函数。它可以很方便地从网站返回的数据中过滤出所需的“有效数据”。

8.3.1 获取百度搜索结果

还是那句老话，想知道 Python 模块最详细的用法，直接用 help 函数就可以了。鉴于 Selenium.Webdriver 的 help 文件太大，分屏显示又不那么方便，干脆将帮助文件保存到文件中慢慢查看。执行命令：

```
python
from selenium import webdriver
import sys
browser = webdriver.PhantomJS()
out = sys.stdout
sys.stdout = open(‘browserHelp.txt’, ‘w’)
help(browser)
sys.stdout.close()
sys.stdout = out
browser.quit()
exit()
```



一定要加上 browser.quit()，否则 cmd.exe 在执行 exit 时会无法退出。

执行结果如图 8-11 所示。

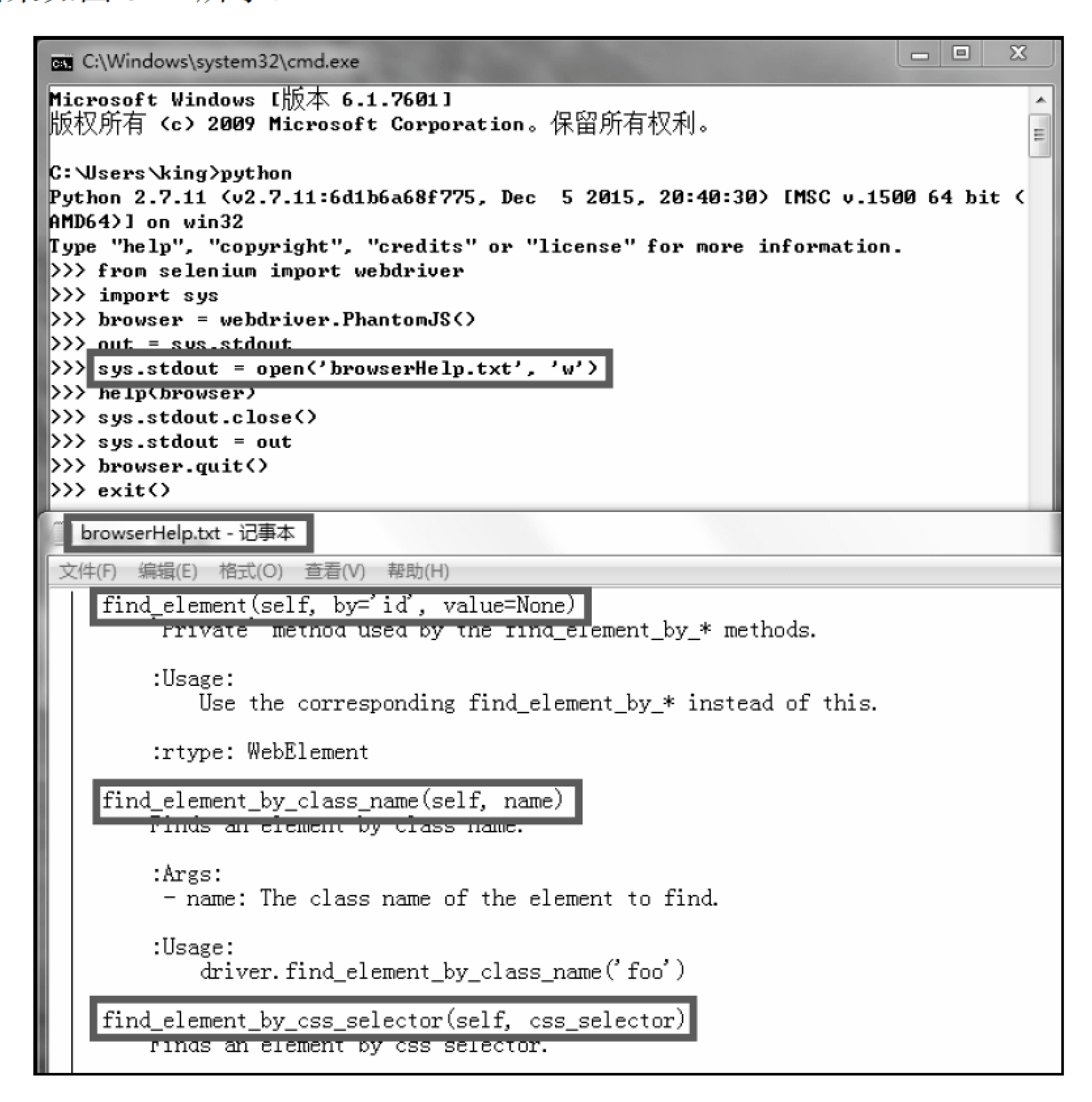


图 8-11　获取 help 文件

想获取“有效信息”，第一步当然是网站获取返回数据，第二步就是定位“有效数据”的位置，第三步就是从定位中获取“有效数据”。

就以百度搜索为例，使用百度搜索“Python Selenium”，并保存第一页搜索结果的标题

和链接。从服务器返回数据，由 PhantomJS 负责，获取返回的数据用 Selenium.Webdriver 自带的方法 page_source，例如：

```
from selenium import webdriver
browser = webdriver.PhantomJS()
browser.get(URL)
html = browser.page_source
```

有两种方法可以得到搜索结果页面。第一种，百度主页还是使用 get 方式上传 request。这里可以先找个浏览器，打开百度后搜索关键词。再把返回来的搜索结果的 URL 保存下来用 Selenium&PhantomJS 打开，再获取返回的数据。第二种，直接用 Selenium&PhantomJS 打开百度的主页，然后模拟搜索关键词。直接从 Selenium&PhantomJS 中返回数据。这里使用第二种方法，可以很清楚地看到 Selenium&PhantomJS 获取数据的过程。

第一步获取搜索结果。打开 cmd.exe，准备好环境。执行命令：

```
python
from selenium import webdriver
browser = webdriver.PhantomJS()
browser.get(‘https://www.baidu.com’)
browser.implicitly_wait(10)
```

执行如图 8-12 所示。

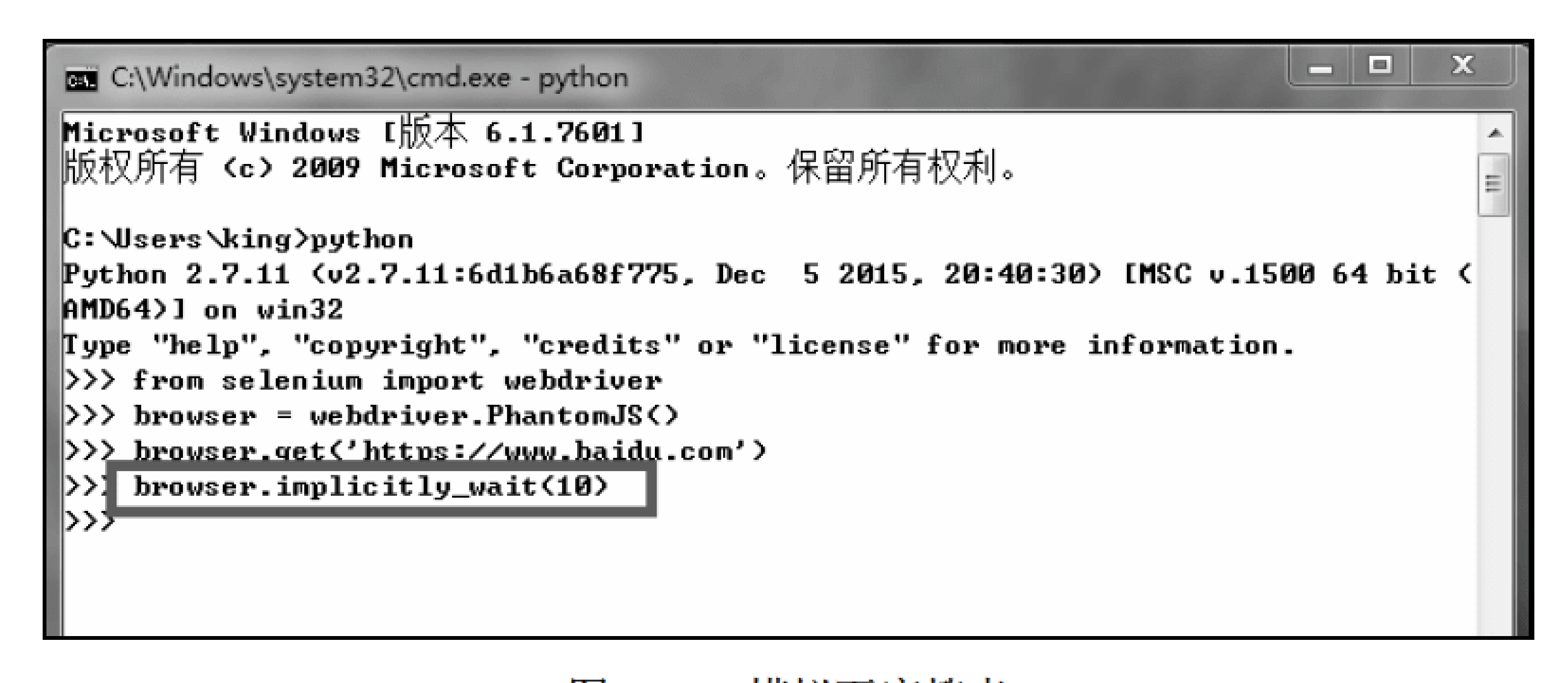


图 8-12　模拟百度搜索

这里要关注一个函数 implicitly_wait()。使用 Selenium&PhantomJS 最大的优势是支持 JavaScript，而 PhantomJS 浏览器解释 JavaScript 是需要时间的。这个时间是多少并不好确定，当然可以用 time.sleep()强行休眠等待一个固定时间。可这个固定的时间定长了，浪费时间；定短了，又没能完整地解释 JavaScript。Implicitly_wait 函数则完美地解决了这个问题，给 implicitly_wait 一个时间参数。Implicitly_wait 会智能等待，只要解释完成了就进行下一步，完全没有浪费时间。下面从网页的框架中选取表单框，并输入搜索的关键词，完成搜索的过程。

8.3.2 获取搜索结果

第二步定位表单框架或“有效数据”位置，可以用 import 导入 bs4 来完成，也可以用 Selenium 本身自带的函数来完成。Selenium 本身给出了 18 个函数，总共 8 种方法从返回数据中定位“有效数据”位置。这些函数分别是：

```
find_element(self, by='id', value=None)
find_element_by_class_name(self, name)
find_element_by_css_selector(self, css_selector)
find_element_by_id(self, id_)
find_element_by_link_text(self, link_text)
find_element_by_name(self, name)
find_element_by_partial_link_text(self, link_text)
find_element_by_tag_name(self, name)
find_element_by_xpath(self, xpath)

find_elements(self, by='id', value=None)
find_elements_by_class_name(self, name)
find_elements_by_css_selector(self, css_selector)
find_elements_by_id(self, id_)
find_elements_by_link_text(self, text)
find_elements_by_name(self, name)
find_elements_by_partial_link_text(self, link_text)
find_elements_by_tag_name(self, name)
find_elements_by_xpath(self, xpath)
```

这 18 个函数前面的 9 个带 element 的函数将返回第一个符合参数要求的 element，后面 9 个带 elements 的函数将返回一个列表，列表中包含所有符合参数要求的 element。命名是 9 个函数，为什么只有 8 种方法呢？上面函数中，不带 by 的函数，配合参数可以替代其他的函数。例如：find_element(by='id', value='abc') 就可以替代 find_element_by_id('abc')。同理 find_elements(by='id', value='abc')也可以替代 find_elements_by_id('abc')。

这 8 种定位方法组合应用，灵活配合，可以获取定位数据中的任何位置。个人觉得，在使用浏览器请求数据时，用 find_element_by_name、find_element_by_class_name、find_element_by_id、find_element_by_tag_name 会比较方便。一般的表单、元素都会有 name、class、id，这样定位会比较方便。如果仅仅是为了获取“有效数据”的位置，那还是 find_element_by_xpath 和 find_element_by_css 比较方便。强烈推荐 find_element_by_xpath，真的是超级方便。

先定位文本框，输入搜索关键词并向服务器发送数据。在 Chrome 中打开百度主页，查看源代码页面（如果想全程无 GUI，也可以直接在 Selenium 中用 page_source 获取页面代码，保存后再慢慢地搜索，不过这样就比较麻烦了）。在源代码页面搜索 type=text，也就是查找页面使用的文本框，搜索结果如图 8-13 所示。



图 8-13　搜索文本框位置

从图 8-13 可以看出文本框里有 class、name、id 属性，可以使用 find_element_by_class_name、find_element_by_id、find_element_by_name 来定位。执行命令：

```
textElement = browser.find_element_by_class_name('s_ipt')
textElement = browser.find_element_by_id('kw')
textElement = browser.find_element_by_name('wd') #这三个任选其一都可以

textElement.clear()
textElement.send_keys('Python selenium')
```

回到 Chorme 中百度源代码页面，搜索 type=submit，定位 submit 按键位置，如图 8-14 所示。



图 8-14　搜索 submit 按键

从图 8-14 可看出，submit 按键有 id、class 属性，可以用 find_element_by_class_name 和 find_element_by_id 定位。执行命令：

```
submitElement = browser.find_element_by_class_name('btn self-btn bg s_btn')
submitElement = browser.find_element_by_id('su')  #这两个任选一个
submitElement.click()

print browser.title
```

执行结果如图 8-15 所示。

```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\king>python
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from selenium import webdriver
>>> browser = webdriver.PhantomJS()
>>> browser.get('https://www.baidu.com')
>>> browser.implicitly_wait(10)
>>>
>>>
>>> textElement = browser.find_element_by_class_name('s_ipt')
>>> textElement.send_keys('Python selenium')
>>>
>>>
>>> submitElement = browser.find_element_by_id('su')
>>> submitElement.click()
>>>
>>>
>>> print browser.title
Python selenium_百度搜索
>>>
```

图 8-15　获取搜索结果

此时 browser 已经获取到了搜索的结果了。

8.3.3　获取有效数据位置

第三步获取“有效数据”位置或者说是 element。先定位搜索结果的标题和链接。回到 Chrome 浏览器，在百度中搜索 Pyton selenium，在搜索结果页面中查看源代码。因为 Chrome 浏览器和 PhantomJS 浏览器返回的结果可能有所不同，这里只需要知道返回结果的大致结构，不需要完全一致。Chrome 浏览器返回结果如图 8-16 所示。



图 8-16　Chrome 浏览器搜索结果

打开源代码页面搜索第一个结果的标题 Selenium with Python，如图 8-17 所示。



图 8-17　搜索结果定位

在这里发现了一个比较特别的属性 class="c-tools"，在代码中查找这个属性，如图 8-18 所示。



图 8-18　搜索 class 属性

发现共有 10 个结果，并且第二个搜索结果的标题和搜索页面中第二个搜索结果相同，再数一数百度搜索结果页面中总共 10 个结果。可以确定所有的搜索结果中都包含有 class="c-tools"标签，可以用 find_elements_by_class_name 定位所有的搜索结果了。执行命令：

```
resultElements = browser.find_elements_by_class_name('c-tools')
len(resultElements)
```

执行结果如图 8-19 所示。

```
C:\Users\king>python
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from selenium import webdriver
>>> browser = webdriver.PhantomJS()
>>> browser.get('https://www.baidu.com')
>>> browser.implicitly_wait(10)
>>>
>>>
>>> textElement = browser.find_element_by_class_name('s_ipt')
>>> textElement.send_keys('Python selenium')
>>>
>>>
>>> submitElement = browser.find_element_by_id('su')
>>> submitElement.click()
>>>
>>>
>>> print browser.title
Python selenium_百度搜索
>>>
>>> resultElements = browser.find_elements_by_class_name('c-tools')
>>> len(resultElements)
10
>>>
```

图 8-19　定位搜索结果



这里使用的是 find_elements，不是 find_element。定位多个结果时用 elements。

一般来说定位结果用 find_element_by_xpath 或 find_element_by_css 比较方便，但如果结果中有特殊的属性，用 find_element_by_class_name 也挺好，哪个方便就用哪一个。

8.3.4　从位置中获取有效数据

有效数据的位置确定后，如何从位置中过滤出有效的数据呢？一般就是获取 element 的文字或者获取 Element 中某个属性值。Selenium 有自己独特的方法，分别是：

```
element.text

element.get_attribute(name)
```

回到 Chrome 浏览器搜索结果的源代码页面，如图 8-20 所示。

```
view-source:https://www.baidu.com/s?wd=python%20selenium&rsv_spt=1
<div class="c-abstract c-abstract-en"><em>Selenium</em> <em>Py          class="c-tools"   第 1 条，共 10 条
Installation 2. Getting Started 3. Navigating 4. Locating Elements 5. Waits 6. Page Objects 7.
WebDriver API 8...</div><div class="f13"><a target="_blank" href="http://www.baidu.com/link?
url=N2Zv2Gbm-PgGppZdZCUOF61vJE3jvz0US-LAWBdShKNpcjEzWD-k1ykdaXZW9xGsjl6bf312ErCR_pcHZrUeX_" class="c-
showurl" style="text-decoration:none;">https://<b>selenium</b>-<b>python</b>.readth... </a><div
class="c-tools" id="tools_4046434806260270696_1" data-tools='{"title":"Selenium with Python — Selenium
Python Bindings 2 ...","url":"http://www.baidu.com/link?url=N2Zv2Gbm-PgGppZdZCUOF61vJE3jvz0US-
LAWBdShKNpcjEzWD-k1ykdaXZW9xGsjl6bf312ErCR_pcHZrUeX_"}'><a class="c-tip-icon"><i class="c-icon c-icon-
```

图 8-20　有效数据

所需的有效数据就是 data-tools 属性的值。执行命令：

```
value = resultElements[0].get_attribute('data-tools')
valueDic = eval(value)
```

```
print valueDic.get('title').decode('utf8')
print valueDic.get('url')
```

执行结果如图 8-21 所示。

```
>>>
>>> value = resultElements[0].get_attribute('data-tools')
>>> valueDic = eval(value)
>>> print valueDic.get('title').decode('utf8')
Selenium with Python — Selenium Python Bindings 2 ...
>>> print valueDic.get('url')
http://www.baidu.com/link?url=6Te0ygsF8h9vDPrYMImuVfItxjOgM0iBpxGMPxpE1rcekZgzUt
Ar7Fwu1DfBJeO3wpTjZKRra3QG2Yp_YVeLga
>>>
```

图 8-21　获取有效数据

遍历 resultElements 列表，可以获取所有搜索结果的 title 和 url。至此，已将 Selenium&PhantomJS 爬虫运行了一遍。根据这个过程可以编写一个完整的爬虫。

8.4 Selenium&PhantomJS 实战一：获取代理

用 Selenium&PhantomJS 完成的网络爬虫，最适合使用的情形是爬取有 JavaScript 的网站，但用来爬其他的站点也一样给力。在 Scrapy 爬虫中曾爬取过代理服务器的例子，这里再以 Selenium&PhantomJS 爬取代理服务器为示例，比较两者有什么不同。

8.4.1 准备环境

在 Scrapy 爬虫中获取了代理，需要自行验证代理是否可用。这次将在 www.kuaidaili.com 中获取已经验证好了的代理服务器。打开目标站点主页，如图 8-22 所示。



图 8-22　目标主页

最终需要获取的有效数据就是代理服务器。从图中可以看出网站也给出了 API 接口。从好的方面想，有现成的 API 接口获取代理服务器会更加方便；但从坏的方面考虑，因为本身就有 API 接口，那么限制爬虫恐怕就更加厉害了。

单击 API 接口的链接查看一下，如图 8-23 所示。



图 8-23　API 限制条件

还好，限制的条件不多，无须添加复杂的反爬虫。下面准备爬虫项目环境，打开 Putty，连接登录到 Linux，进入爬虫项目目录，执行命令：

```
cd code/crawler
mkdir seleniumProject
cd seleniumProject
```

执行结果如图 8-24 所示。

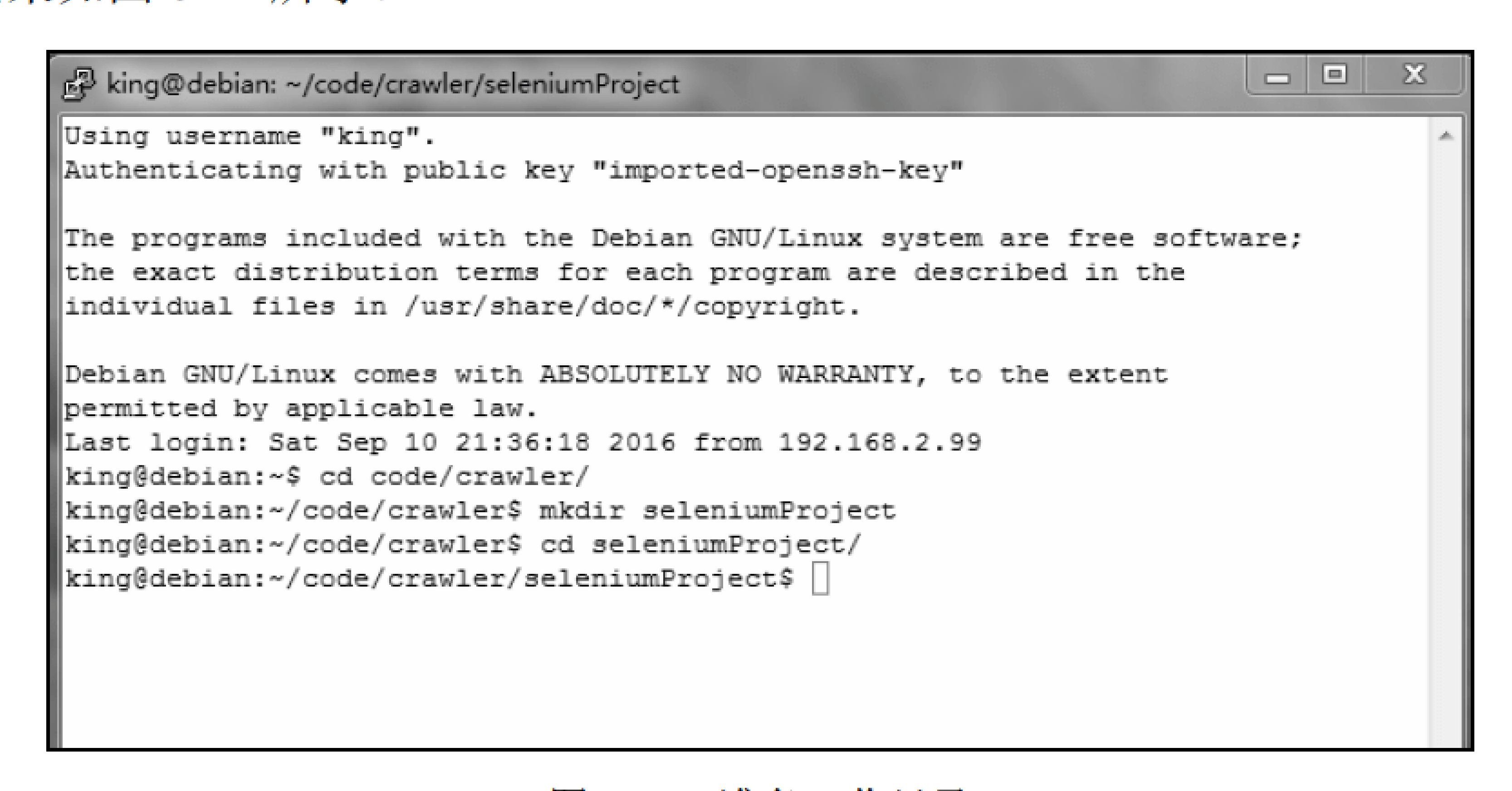


图 8-24　准备工作目录

下面就可以在该目录下编写爬虫文件 getProxyFromKuaidaili.py。

8.4.2　爬虫代码

【示例 8-1】getProxyFromKuaidaili.py 的代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
__author__ = 'hstking hstking@hotmail.com'


from selenium import webdriver
from myLog import MyLog as mylog


class Item(object):
    ip = None  #代理 IP
    port = None  #代理端口
    anonymous = None  #是否匿名
    type = None  #类型
    support = None  #支持的协议
    local = None  #物理地址
    speed = None  #代理速度

class GetProxy(object):
    def __init__(self):
        self.startUrl = 'http://www.kuaidaili.com/proxylist/'
        self.log = mylog()
        self.urls = self.getUrls()
        self.proxyList = self.getProxyList(self.urls)
        self.fileName = 'proxy.txt'
        self.saveFile(self.fileName, self.proxyList)

    def getUrls(self):
        urls = []
        for i in xrange(1,11):
            url = self.startUrl + str(i)
            urls.append(url)
            self.log.info('get url %s to urls' %url)
        return urls

    def getProxyList(self, urls):
        browser = webdriver.PhantomJS()
        proxyList = []
        item = Item()
        for url in urls:
            browser.get(url)
            browser.implicitly_wait(5)
            elements = browser.find_elements_by_xpath('//tbody/tr')
            for element in elements:
                item.ip =
element.find_element_by_xpath('./td[1]').text.encode('utf8')
                item.port =
```

```
element.find_element_by_xpath('./td[2]').text.encode('utf8')
                item.anonymous =
element.find_element_by_xpath('./td[3]').text.encode('utf8')
                item.type =
element.find_element_by_xpath('./td[4]').text.encode('utf8')
                item.support =
element.find_element_by_xpath('./td[5]').text.encode('utf8')
                item.local =
element.find_element_by_xpath('./td[6]').text.encode('utf8')
                item.speed =
element.find_element_by_xpath('./td[7]').text.encode('utf8')
                proxyList.append(item)
                self.log.info('add proxy %s:%s to list' %(item.ip,
item.port))
        browser.quit()
        return proxyList

    def saveFile(self, fileName, proxyList):
        self.log.info('add all proxy to %s' %fileName)
        with open(fileName, 'w') as fp:
            for item in proxyList:
                fp.write(item.ip + '\t')
                fp.write(item.port + '\t')
                fp.write(item.anonymous + '\t')
                fp.write(item.type + '\t')
                fp.write(item.support + '\t')
                fp.write(item.local + '\t')
                fp.write(item.speed + '\n')


if __name__ == '__main__':
    GP = GetProxy()
```

按:wq 保存结果，再将之前项目中用到过的 myLog.py 复制到当前目录下。查看当前目录，执行命令：

```
tree
```

执行结果如图 8-25 所示。

```
king@debian:~/code/crawler/seleniumProject$
king@debian:~/code/crawler/seleniumProject$ tree
.
├── getProxyFromKuaidaili.py
└── myLog.py

0 directories, 2 files
king@debian:~/code/crawler/seleniumProject$
```

图 8-25　显示目录文件

运行爬虫文件，执行命令：

```
python getProxyFromKuaidaili.py
tree
```

执行结果如图 8-26 所示。

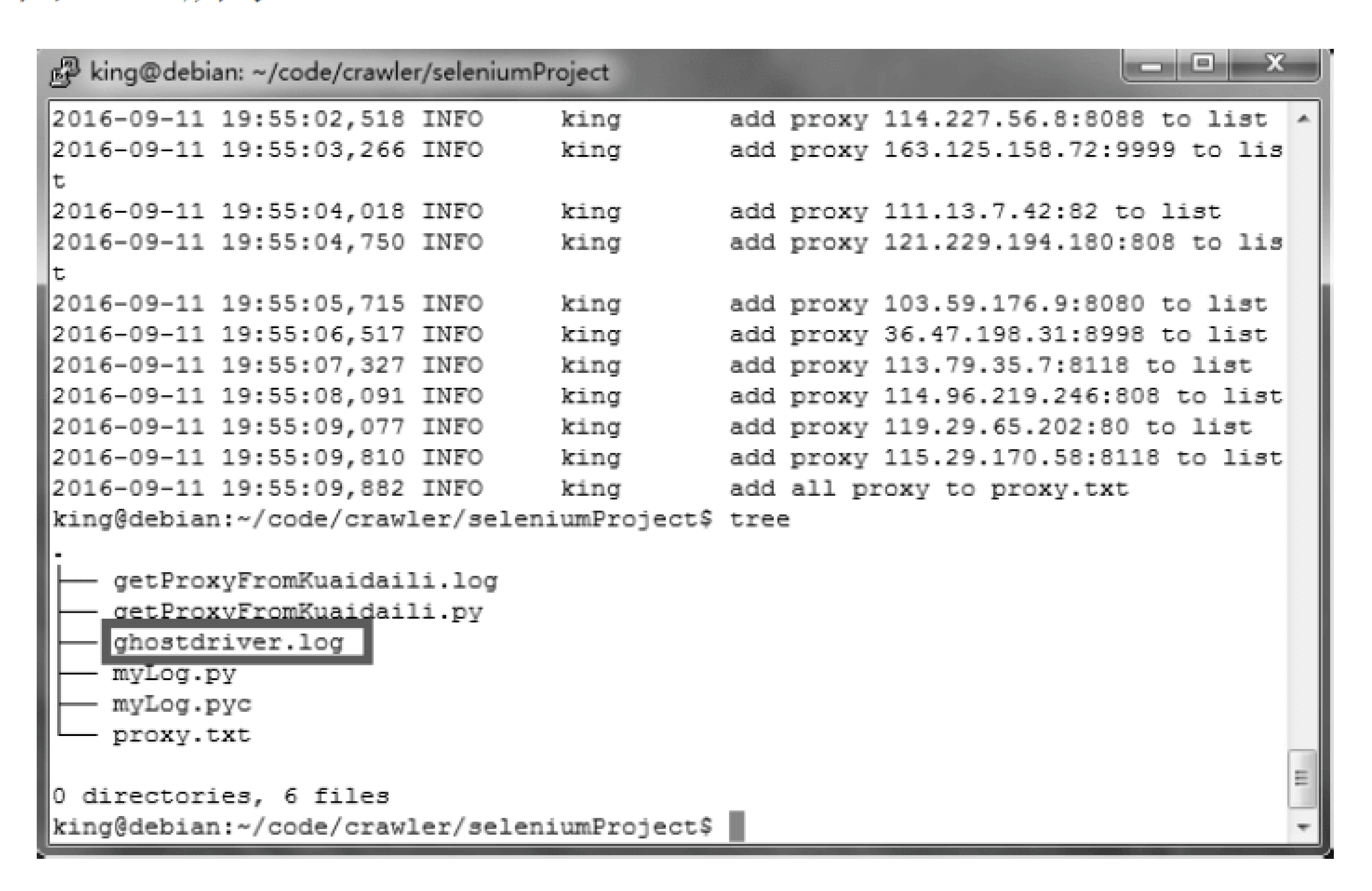


图 8-26　运行爬虫

这里的 getProxyFromKuaidaili.log 是用户定义的日志文件。Proxy.txt 是最终得到的结果。Ghostdriver.log 是运行 PhantomJS 的日志文件。

8.4.3　代码解释

示例 8-1 这个爬虫程序本身并不复杂。第 6~7 行是导入所需的模块，其中 myLog 模块是自定义模块，也就是后来复制到当前目录的 myLog.py 文件。

第 10~17 行定义了一个 Item 类。这个类的作用是为了方便装载爬虫获取的数据，基本包含了网页中的所有项。

第 19~67 行定义了一个从 kuaidili 站点中获取 proxy 的类。这个类包含了 3 个类函数。getUrls 函数用于返回一个列表，这个列表包含了所有有效数据的网页地址。getProxyList 函数，从网页中获取有效数据，并保存到一个列表中。最后的 saveFile 函数，将所有列表中的数据保存到文件中。

8.5　Selenium&PhantomJS 实战二：漫画爬虫

Selenium&PhantomJS 可以说专为 JavaScript 而生。从前面的项目中已经熟悉了 Selenium&PhantomJS 的用法。下面开始用 Selenium&PhantomJS 获取 JavaScript 返回的数据。一般来说，网站上用 JavaScript 返回数据，主要是为了美观，第二目的估计就是为了增加爬虫的难度。这里只是讨论技术实现的手段，请遵循“不作恶”原则，不要侵犯他人的知识产权。

8.5.1 准备环境

一般来说在线看漫画的网站都会使用 JavaScript 来返回页面。在 Chrome 中打开百度搜索，搜索在线漫画，如图 8-27 所示。



图 8-27 寻找目标站点

第一个搜索结果已经很明确地提出了“禁止下载”，那就算了，就找第二个好了。打开第二个搜索结果，如图 8-28 所示。



图 8-28 选取目标

任选一个目标都可以，这里选取第一个漫画。打开漫画浏览页面，如图 8-29 所示。

图 8-29　爬虫起始页面

这个爬虫将在 Windows 下使用 Eclipse 完成。打开 Eclipse，新建 PyDev Project，项目名为 getCartoon。

8.5.2　爬虫代码

在 getCartoon 项目中创建一个 PyDev Module，名字为 cartoon1.py。

【示例 8-2】cartoon1.py 的代码如下：

```
#!/usr/bin/evn python
#-*- coding: utf-8 -*-
'''
Created on 2016 年 9 月 10 日

@author: hstking hstking@hotmail.com
'''

from selenium import webdriver
from mylog import MyLog as mylog
import os
import time

class GetCartoon(object):
    def __init__(self):
        self.startUrl = u'http://www.1kkk.com/ch1-406302/'
        self.log = mylog()
        self.browser = self.getBrowser()
```

```
        self.saveCartoon(self.browser)


    def getBrowser(self):
        browser = webdriver.PhantomJS()
        try:
            browser.get(self.startUrl)
        except:
            mylog.error('open the %s failed' %self.startUrl)
        browser.implicitly_wait(20)
        return browser

    def saveCartoon(self, browser):
        cartoonTitle = browser.title.split('_')[0]
        self.createDir(cartoonTitle)
        os.chdir(cartoonTitle)
        sumPage = int(self.browser.find_element_by_xpath('//font[@class="zf40"]/span[2]').text)
        i = 1
        while i<=sumPage:
            imgName = str(i) + '.png'
            browser.get_screenshot_as_file(imgName)
            self.log.info('save img %s' %imgName)
            i += 1
            NextTag = browser.find_element_by_id('next')
            NextTag.click()
#             browser.implicitly_wait(20)
            time.sleep(5)
        self.log.info('save img sccess')
        exit()

    def createDir(self, dirName):
        if os.path.exists(dirName):
            self.log.error('create directory %s failed, hava a same name file or directory' %dirName)
        else:
            try:
                os.makedirs(dirName)
            except:
                self.log.error('create directory %s failed' %dirName)
            else:
                self.log.info('create directory %s success' %dirName)


if __name__ == '__main__':
    GC = GetCartoon()
```

再将之前项目中的 mylog.py 复制到项目中（Linux 中复制的是 myLog.py，实际上是一个文件，只不过 Windows 下不区分大小写而已）。单击 Eclipse 图标栏的运行图标，执行结果如图 8-30 所示。

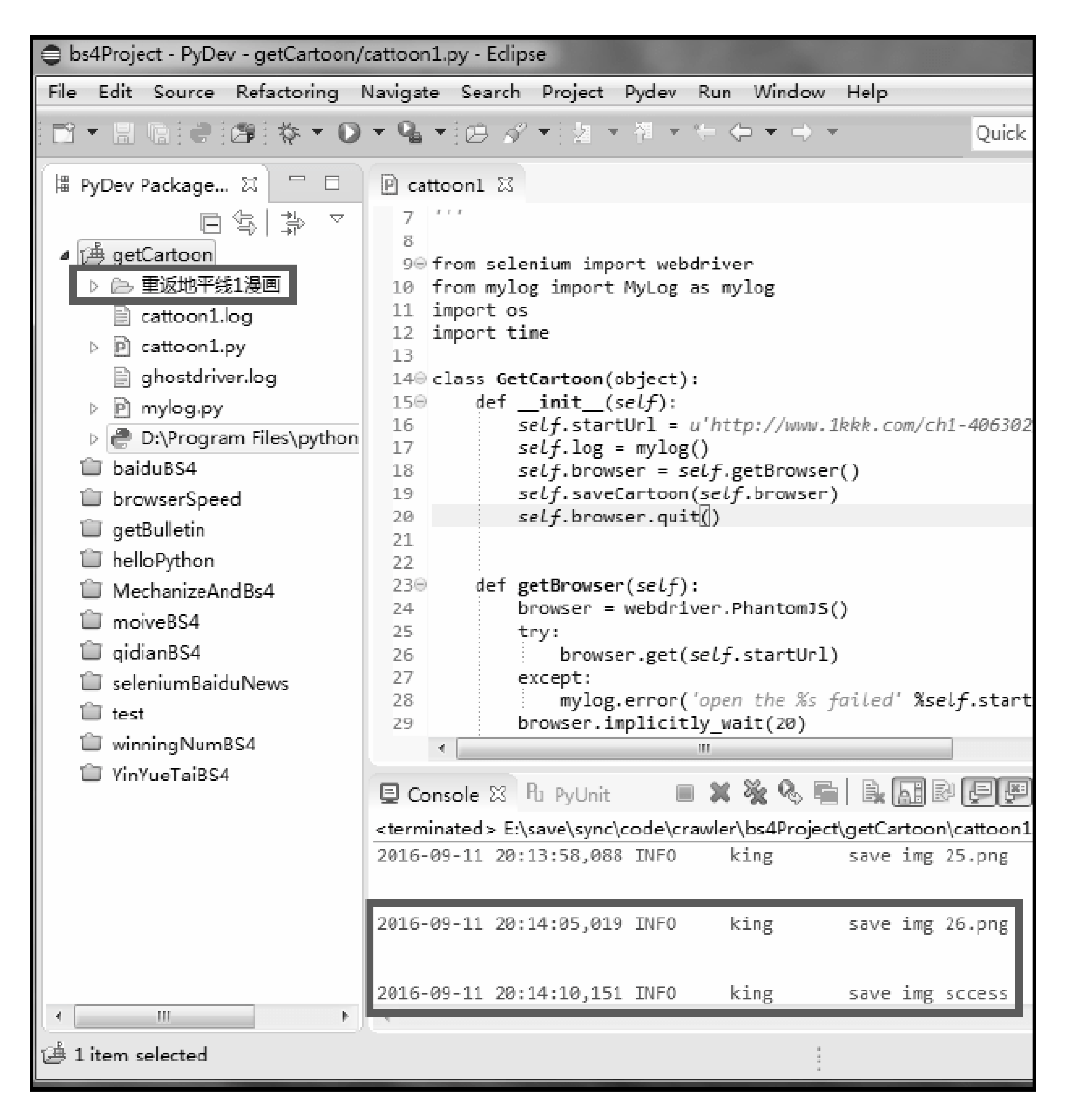


图 8-30　运行爬虫 getCartoon1

日志文件显示操作成功。项目中也得到了漫画的目录。打开下载的漫画目录，如图 8-31 所示。



图 8-31　获取的结果

最终保存的结果不是单纯的漫画，而是整个页面的截图。

8.5.3 代码解释

示例 8-2 这个爬虫不长，才 60 多行。第 9~12 行是导入所需的模块。第 14~58 行是爬虫类，这个爬虫类只有 3 个类函数。

第 22~29 行是类函数 getBrowser 的原型，它的作用是返回一个 browser 对象。这里稍微要注意点的就是 browser.implicitly_wait(20)，它的作用设定了智能等待的最长时间。

第 31~47 行是类函数 savaCartoon。这个函数将从网站中获取图片，并保存到新建立的文件夹中。文件夹的名字是从网页的 title 中获取的。从网页中获取了这个漫画的总页数，因为这个漫画在最后一页（第 26 页）还是有“下一页”的按钮。没办法通过是否存在“下一页”按钮来确定是不是最后一页。所以必须得先获取这个漫画的总页数。Seleniuim&PhantomJS 解释了页面的 JavaScript，也将解释后得到的图片显示在浏览器上了。但这个站点在防盗链上做得很到位，只要在页面上执行一次刷新操作，网站就判为盗链，显示出防止盗链的图片出来，并且得到的图片链接地址也无法下载，所以最简单的方法就是对整个页面进行截图。好在 Selenium 本身就有截图工具，还算比较方便。另外，在 NextTag.click()之后使用的并不是 Selenium 的智能等待 implicitly_wait，而是 time.sleep。是因为 implicitly_wait 对 NextTag.click()并不起作用，反而使用 time.sleep 的效果却很不错。

第 49~58 行是类函数 createDir，作用是创建一个目录。为了防止有同名的目录，先在函数内作出判断。

这个爬虫虽然将漫画全部保存下来了，但也把页面上多余的部分保存了。略有瑕疵，如果想追求完美，也可以通过其他的模块将所需的漫画裁剪下来。

8.6 本章总结

Selenium&PhantomJS 爬虫功能很强，但效率上并不高。对访问者限制不严格的网站，一般不建议使用 Selenium&PhantomJS 爬虫。仅仅是对那些通过 JavaScript 返回有效数据的网站，Selenium&PhantomJS 爬虫效果还不错，也可能是最好的选择了。